AGENT EVALUATION
SCIENCE

Can Small Databases Preserve SQL Evaluation Signals?

Kate Gwimm[1] Carson Eisenach[1]
[1] Amazon

Abstract

Production AI agents that act on real systems often need to query relational data by writing SQL queries. In enterprise settings, tables are often very large, so benchmarking such agents and giving feedback during training is both expensive and slow. A sandbox, a small database built for a single gold query, can make such executions fast, but how faithfully it represents the warehouse depends on its constraints, how it is built, and which properties of the data it aims to preserve. Past work builds counterexamples that separate a pair of queries, test suites of distilled synthetic databases selected to maximize coverage over a gold query, or reduced copies of an existing database that keep the gold query's coverage. We target warehouse-referenced fidelity instead: whether a sandbox gives new attempts at a query the evaluation signal of one particular warehouse, under different downstream scoring rules and practical budgets.

We (1) pose the problem of what a small warehouse subset should preserve to support evaluating unseen SQL attempts, (2) formulate an evaluation framework whose measurement of closeness depends on the downstream task, and (3) compare three construction methods for 55 gold queries at three budgets under three measurements. We find that the rows that are chosen matter more than the budget: at 20k rows per table, a rule-based lineage + traps sandbox agrees with the warehouse's verdict on 92% of attempts, against 66% for join-aware uniform sampling, and accepts 13% of the attempts the warehouse rejects, against 52%. With 100 times the budget, join-aware uniform sampling agrees on 79% and accepts 33%. An ablation attributes most of the overall gain to lineage; traps mainly improve rejection of wrong attempts that share rows with the gold. Adding rows with an LLM proposer and critic lowers the false-accept rate at 20k to 7%. Despite high agreement with the warehouse's verdicts, these sandboxes overstate partial credit for wrong attempts close to the gold at every budget tested, making their graded scores unreliable as reward signals.

1 Introduction

Production tables in enterprise settings can hold hundreds of billions of rows. Evaluating data agents by their execution accuracy on a benchmark (Yu et al., 2018; Lei et al., 2025) (the share of questions on which an agent's query returns the reference query's output) or serving a query tool in agent training at scale often requires multiple attempts per question, which is infeasible on full data.

Data warehouses also supply context that a schema alone cannot capture. Documentation can be stale, constraints can be incompletely declared, and meaningful relationships among tables may be implicit. Synthesized databases (Binnig et al., 2007; Chandra et al., 2015) and distilled test suites (Zhong et al., 2020) evaluate attempts on databases constructed around the gold query; their target is semantic correctness across databases, not agreement with the evaluation signal of a particular warehouse, whose undocumented semantics they do not see. The closest predecessor, QAShrink (Tuya et al., 2009, 2016), reduces an existing database while preserving the gold query's coverage, and evaluates the reduced database's

fault-detection ability. We adapt it as our rule-based baseline and ask a different question: whether a sandbox of real warehouse rows preserves the warehouse's evaluation signal for new attempts, unseen during construction, under a per-table budget. What it should preserve depends on the downstream application of the sandbox. We investigate two evaluation signals: the warehouse's accept/reject verdict and its graded score, which measures an attempt's row similarity to the gold.

We first introduce an evaluation framework that supports different definitions of fidelity. Construction of the sandbox requires queries to the full warehouse, a cost that pays off only when it is amortized over many later executions; we report the time of those executions, not the cost of construction. The trade-off between sandbox size and fidelity depends on the application, such as whether some error is acceptable and how that error is computed.

On 55 gold queries over 62 tables, each reading a table larger than any of our budgets, we compare three sandbox construction methods: uniform sampling of join keys (join-aware uniform), a rule-based method (lineage + traps) and an LLM proposer and critic on top of it (proposer + critic). We measure whether a sandbox's verdict on nearby attempts (queries written to answer the same question as a gold query, such as model answers and plausible mistakes) agrees with the warehouse's, and whether the attempts' row similarity to the gold on the sandbox matches their row similarity to the gold on the warehouse.

2 Definitions and evaluation framework

Fix a gold query q^* and a warehouse reference $W = \{T_1, \ldots, T_m\}$ containing its source tables, with any task-specific date windows fixed in advance. A sandbox suite contains k instances:

$$D = \{D_1, \ldots, D_k\}, \qquad D_i = \{T_1^{(i)}, \ldots, T_m^{(i)}\}, \qquad T_j^{(i)} \subseteq T_j, \quad |T_j^{(i)}| \le B.$$

Each instance preserves the table schemas and holds at most B rows per table. The suite therefore has a total allowance of kB rows per source table. Let $\tilde{q}$ be an attempt to answer the same question as q^*, drawn from a population that the builder does not observe during construction. Writing $E(q, X)$ for execution of q on database X, define

$$y_i = E(\tilde{q}, D_i), \quad y_i^* = E(q^*, D_i), \qquad y_W = E(\tilde{q}, W), \quad y_W^* = E(q^*, W).$$

Outputs are multisets of rows, with cells compared exactly. An execution error is unequal to every output, including another error, and has row similarity zero.

A downstream task chooses a per-instance measurement f and an aggregation rule f_{agg}:

$$s_i = f(y_i, y_i^*, y_W, y_W^*), \qquad F(\tilde{q}, D, W) = f_{\text{agg}}(s_1, \ldots, s_k).$$

The evaluation estimates this score over held-out attempts at each construction budget. Its meaning depends on the chosen measurement; the following three choices assess a sandbox used to evaluate SQL.

Verdict agreement. An attempt passes on an instance when it returns the gold's output there:

$$p_i = \mathbf{1}[y_i = y_i^*], \qquad \ell = \mathbf{1}[y_W = y_W^*], \qquad F_{\mathsf{V}} = \mathbf{1}\left[\min_i p_i = \ell\right].$$

The suite accepts only when every instance passes the attempt. A majority vote is another possible rule for a suite and must be chosen before evaluation. We report the mean of F_{V} separately on warehouse-wrong attempts ($\ell = 0$, one minus the false-accept rate) and warehouse-correct attempts ($\ell = 1$, one minus the false-reject rate). Correctness here means agreement with the gold on the fixed reference W.

Row-similarity agreement. For result multisets A and R, let $g(A, R) = |A \cap R|/|A \cup R|$ be their Jaccard similarity. Set $g(\varnothing, \varnothing) = 1$. Compare the attempt with the gold within each database,

then measure how closely the sandbox reproduces the warehouse's similarity score:

$$a_W = g(y_W, y_W^*), \qquad a_D = \frac{1}{k}\sum_i g(y_i, y_i^*), \qquad F_{\text{J}} = 1 - |a_D - a_W|.$$

Thus $F_{\text{J}} = 1$ means the attempt receives the same similarity score on the sandbox and the warehouse.

Agreement within τ. The sandbox and the warehouse must agree on whether the attempt is within τ of the gold, with instances combined as for the verdict:

$$p_i^\tau = \mathbf{1}[g(y_i, y_i^*) \geq 1 - \tau], \qquad \ell^\tau = \mathbf{1}[a_W \geq 1 - \tau], \qquad F_\tau = \mathbf{1}\left[\min_i p_i^\tau = \ell^\tau\right],$$

so $F_0 = F_{\text{V}}$. The verdict and agreement within τ require every instance to pass, while row similarity averages over instances.

All our experiments use one instance ($k = 1$), so the aggregation choices described above do not affect our results. The framework also supports other measurements, such as comparing a query's output on the sandbox directly with its output on the warehouse.

3 Experiment

3.1 Setting

Gold queries come from an internal text-to-SQL dataset, restricted to tasks whose gold result is reproducible and nonempty. Each gold query reads one to five Redshift tables, which form W. For each task, we build the three sandboxes described below at three budgets (20k, 200k, 2M rows per table), each a single sandbox instance. Of the 65 tasks on which every method was built at every budget, we report the 55 with a table of more than 2M rows in W, which no budget holds whole; on the other 10 (at most 1.1 million rows per table), every method agrees with the warehouse's verdict on 301 of 304 attempts at 20k and on all of them at 200k and 2M. On 8 of the 55 (local tasks), W is a DuckDB copy of those tables; on the other 47 (remote tasks), reference outputs for the gold and attempt queries come from executing them against the source tables in Redshift.

Held-out attempts consist of distinct model answers from earlier evaluation runs and up to eight plausible mistakes written by Claude Opus 4.6 without seeing a sandbox. These sets are fixed across methods and budgets. Labels come from agreement with the gold on W. On those 55 tasks, the fidelity results reported cover 1,088 attempts (730 model answers, 358 LLM-written), of which 397 are correct on W (on 51 tasks) and 691 wrong; 294 of the wrong ones (on 49 tasks) share a row with the gold on W, and 397 share none. Scores are averaged within each task and then across tasks. Intervals are 95% bias-corrected and accelerated bootstrap intervals from 20,000 resamples of the tasks. A p-value for a difference between two methods comes from a paired sign-flip test over tasks. When only a few tasks differ between two methods, the interval for the difference can exclude 0 while the test does not reject, since a resample rarely misses every task that differs; a difference is claimed only from the test.

Verdict agreement is scored on all 1,088 attempts, while the other two measurements are scored on the 294 attempts that are wrong but share a row with the gold on the warehouse. These are the attempts that earn partial credit on the warehouse; we do not report row-similarity agreement or agreement within τ on correct attempts or on wrong ones that share no row with the gold, because there the three measurements nearly coincide (Appendix B).

We do not score how well a sandbox reproduces an attempt's own warehouse output: a sandbox holds only some of the rows, so that score mostly tracks the budget.

3.2 Methods

We compare three construction methods (Appendix A):

- **Join-aware uniform** samples join-key values without using the gold query's conditions. It orders the values pseudo-randomly by their hashes, then selects as many as the per-table budget of B rows allows. For each selected value, it keeps all rows with that value in every table joined on the key. This preserves all matching rows for the selected values. For tables joined on a different key, it keeps rows that match those already selected. Any table with at most B rows is kept whole. Sampling each table independently would keep few rows that still join (Chaudhuri et al., 1999). On 23 remote tasks, join-aware uniform built no sandbox at some budget, because pulling a table's key values from Redshift failed or timed out; there we sample each table's rows uniformly instead, at every budget. On the 24 remote tasks where both were built, the two had similar agreement with the warehouse's verdict (0.63 and 0.65 at 20k, 0.81 and 0.80 at 2M).
- **Lineage + traps** first selects source rows that satisfy the gold query's filters and joins. It then adds "trap" rows that the gold excludes but an incorrect query could include, together with the rows needed to join them. For example, if the gold requires A `AND` B, the method looks for rows that satisfy B but not A, or A but not B:

  `(A IS NOT TRUE) AND B` and `A AND (B IS NOT TRUE)`.

  These rows can expose an attempt that omits one of the two conditions. The method seeks up to 20 trap rows per condition. Additional rules select rows to expose mistakes in joins, date boundaries, alternative values, and letter case, and rows that make nonzero contributions to aggregates. Each table is limited to B rows. When a selection is too large, the method samples it by hashing, using a shared join key where available. This adapts coverage-aware database reduction (QAShrink; Tuya et al., 2009, 2016) to warehouse tables with a per-table budget. The budget can exclude rows needed to expose a mistake or preserve an aggregate, so agreement with the warehouse must be measured empirically. It may retain fewer rows than the other two methods.
- **Proposer + critic** starts from the lineage + traps sandbox. An initial Claude Sonnet 4.6 proposer session inspects the data and adds rows through bounded tools to expose additional query mistakes. The sandbox then undergoes up to three refinement rounds. Each round starts with a Claude Opus 4.6 critic proposing five candidate mistakes that the sandbox may still accept. A Sonnet 4.6 judge screens candidates that pass on the sandbox for plausibility without seeing it. For plausible candidates confirmed wrong on W, we trace output differences from the gold back to source rows and add them with their join partners if they expose the error on the sandbox. If a candidate still passes, the proposer gets another tool session to seek distinguishing rows. All additions respect the same per-table budget. We hypothesize that this adaptive search uncovers gaps missed by lineage + traps and improves fidelity on unseen attempts.

Budgets and speedup. On the 47 remote tasks, the tables hold a median of 1.1 billion rows (up to 421 billion); an attempt rerun on Redshift takes a median of 163 s, and 4–36 ms on a sandbox of 20k–2M rows per table (median speedup per attempt 4,400–37,600×, over 605–613 attempts). The speedup reflects both data reduction and differences in engine and execution environment: Redshift reads these tables from S3 at query time, while DuckDB holds a sandbox in memory. To isolate data reduction, we also run W and the sandbox on the same DuckDB host. Memory constraints restrict this comparison to the 8 local tasks (152 attempts), whose largest tables hold 2.2 to 29 million rows, far smaller than the remote warehouses. On these tasks, an attempt takes a median of 41 ms on W and 4.4–5.8 ms on a sandbox of 20k rows per table (median speedup per attempt 6–8×); at 2M rows per table, the speedup is 1.4–2.7×. These local speedups describe warehouses that already fit in memory; this comparison does not isolate the contribution of data reduction at the scale of the remote tasks (Appendix B).

	Join-aware uniform			Lineage + traps			Proposer + critic		
Share of attempts	20k	200k	2M	20k	200k	2M	20k	200k	2M
Agrees with the warehouse	0.660	0.722	0.787	0.917	0.924	0.927	**0.950**	**0.952**	**0.953**
Wrong attempt accepted:									
both return no rows	0.248	0.204	0.155	0.009	0.004	0.015	0.004	0.004	0.004
the attempt returns the gold's rows	0.084	0.065	0.049	0.060	0.059	0.047	0.030	0.034	0.032
Correct attempt rejected	0.008	0.009	0.009	0.014	0.013	0.012	0.016	0.011	0.012
False-accept rate	0.523	0.428	0.332	0.133	0.105	0.099	0.075	0.070	0.065
False-reject rate	0.040	0.045	0.045	0.037	0.036	0.049	0.043	0.030	0.049

Table 1 | Verdict outcomes by method and budget in rows per table, as shares of a task's attempts averaged over tasks, on the same 1,088 attempts on 55 tasks for every method and budget (join-aware uniform: uniform rows on 23 of them). A sandbox agrees with the warehouse when it accepts a correct attempt or rejects a wrong one; the three rows below it are the ways it can disagree, so in each column the four rows above the line add up to 1. The two rows below the line are conditional: the share of a task's wrong attempts that the sandbox accepts (691 attempts) and of its correct attempts that it rejects (397 attempts), averaged over tasks.

3.3 Results

The rows a sandbox holds matter more than how many (Table 1). Every method rejects at most 0.05 of correct attempts, so the methods differ in how many wrong attempts they accept. At 20k rows per table, lineage + traps agrees with the warehouse's verdict on 0.26 more attempts than join-aware uniform (0.19–0.34, $p < 0.001$), and its false-accept rate is 0.13 (0.09–0.21), against 0.52 (0.41–0.63). Join-aware uniform gains with the budget while lineage + traps stays at 0.92–0.93, so the gap shrinks to 0.20 at 200k (0.13–0.29, $p < 0.001$) and 0.14 at 2M (0.06–0.23, $p < 0.001$). Against join-aware uniform at 2M, which holds a median of 206 times its rows, lineage + traps at 20k agrees on 0.13 more attempts (0.07–0.21, $p < 0.001$; 24 tasks one way, 16 the other).

Lineage explains most of the overall gain; traps mainly help on overlapping wrong attempts. Rows meeting the gold's conditions can prevent wrong attempts from passing merely because both outputs are empty. In the ablation at 20k rows per table, lineage only (lineage + traps without its trap rows) retains 0.24 of the 0.26 gain in verdict agreement over join-aware uniform. Traps add 0.024 overall ($p = 0.003$), but raise rejection of wrong attempts that share a row with the gold on W by 0.104 ($p < 0.001$; Appendix B).

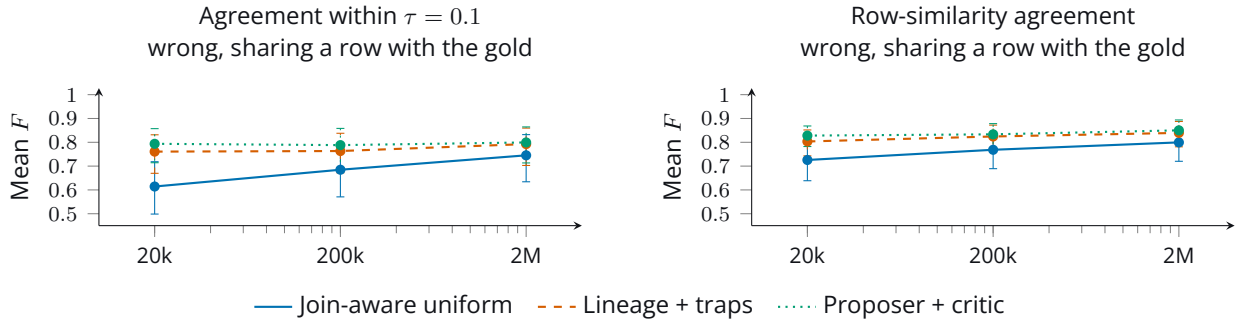


Figure 1 | Fidelity against the budget in rows per table on the 294 attempts (49 tasks) that are wrong but share a row with the gold on the warehouse: agreement within $\tau = 0.1$ and row-similarity agreement. Row-similarity agreement is one minus the gap between the attempt's row similarity to the gold on the sandbox and on the warehouse: 1 means the same score on both, 0.8 means off by 0.2 on average. Bars, 95% task-bootstrap intervals.

Lineage + traps and proposer + critic agree with the warehouse's verdict on most attempts, but overstate partial credit. We call these two the gold-guided sandboxes, since both choose rows by the gold query's conditions. Their pass/fail disagreements with the warehouse are concentrated on wrong attempts that share an output row with the gold on the warehouse. Across the remaining attempts—correct attempts and wrong attempts with no output overlap—verdict agreement is at least 0.94 (Appendix B).

For wrong attempts that share an output row with the gold on the warehouse, the sandboxes also overstate partial credit on average (Figure B2). As the budget increases from 20k to 2M rows per table, the average absolute difference between sandbox and warehouse similarity scores falls from 0.20 to 0.16 for lineage + traps and from 0.17 to 0.15 for proposer + critic (Figure 1). A verdict needs one row that tells an attempt from the gold, while row similarity needs such rows in proportion to the rows the two share. Agreement within $\tau = 0.1$, which neither method was built for, is also lower than their verdict agreement on the same attempts (Figure 1).

Used as a graded training reward, the row similarity of the gold-guided sandboxes would therefore overstate how close these wrong attempts are to the gold.

Proposer + critic adds a small gain over lineage + traps. Proposer + critic's verdict agrees with the warehouse's on more attempts than that of lineage + traps at every budget: 0.03 more at 20k (0.02–0.06, $p < 0.001$, higher on 15 of the 16 tasks where the two differ), 0.03 at 200k (0.02–0.06, $p < 0.001$) and 0.03 at 2M (0.01–0.07, $p = 0.004$). Its false-accept rate at 20k is 0.07, against 0.13 ($p < 0.001$). This gain in verdict agreement comes from rejecting more of the wrong attempts that share a row with the gold (0.86 against 0.75 at 20k, a difference of 0.11, 0.06–0.19, $p < 0.001$).

4 Future work

Additional measurements and hill climbing. The current proposer + critic method hill climbs on verdict agreement: its critic looks for wrong attempts that the sandbox accepts. It could instead hill climb on another measurement by finding attempts whose sandbox and warehouse scores differ—for example, in row similarity to the gold by more than 0.1—and adding rows to reduce the discrepancy within the same budget. A further measurement is whether the sandbox preserves the warehouse's ranking of attempts by row similarity to the gold. If these scores are used to select preferred attempts for training, ranking disagreements would produce different training preferences.

Constructing multiple sandboxes per task. We currently build one sandbox per task for each method and budget. Extending these methods to a set of sandboxes requires algorithms for allocating rows across the set. One approach would build sandboxes sequentially, with each new sandbox targeting wrong attempts that all earlier sandboxes still accept; another would select rows for the set jointly to cover different mistakes. Evaluation would specify how the sandboxes' verdicts and similarity scores are combined, and compare the set with a single sandbox on verdict, similarity and ranking agreement at the same total row budget per table.

Sandboxes in reinforcement learning environments. A sandbox can serve as the query tool and the reward of an environment for training a text-to-SQL agent. The open question is whether training against a sandbox improves the agent's execution accuracy on the warehouse compared to attempts using proxy metrics.

Scope. Our evidence covers one warehouse, one dataset and 55 tasks, with one build of each sandbox; repeated builds would show how much a single construction varies, and other applications (exploration, monitoring) need their own measurements.

LLM Usage Disclosure

We used LLMs in this work, in addition to human effort, to perform more extensive literature reviews, to implement code with tools like Copilot, and to edit the writing of this paper.

References

Carsten Binnig, Donald Kossmann, Eric Lo, and M. Tamer Özsu. QAGen: Generating query-aware test databases. In *SIGMOD*, 2007. Cited at: 1.

Bikash Chandra, Bhupesh Chawda, Biplab Kar, K. V. Maheshwara Reddy, Shetal Shah, and S. Sudarshan. 2015. Data generation for testing and grading SQL queries. *The VLDB Journal*. Cited at: 1.

Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. In *SIGMOD*, 1999. Cited at: 1.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *ICLR*, 2025. Cited at: 1.

Javier Tuya, María José Suárez-Cabal, and Claudio de la Riva. Query-aware shrinking test databases. In *DBTest*, 2009. Cited at: 1, 2.

Javier Tuya, Claudio de la Riva, María José Suárez-Cabal, and Raquel Blanco. 2016. Coverage-aware test database reduction. *IEEE Transactions on Software Engineering*. Cited at: 1, 2.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *EMNLP*, 2018. Cited at: 1.

Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-SQL with distilled test suites. In *EMNLP*, 2020. Cited at: 1.

A Construction methods

Figure A1 shows how the three sandboxes are built.

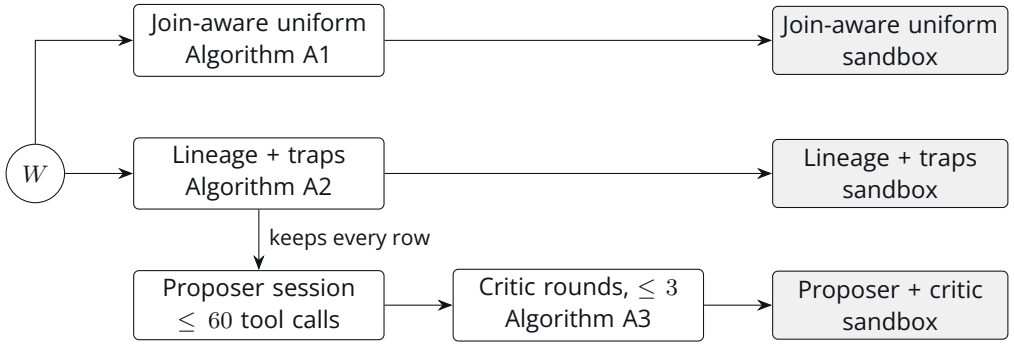


Figure A1 | How the three sandboxes are built at each budget B, from real rows of W only.

A.1 Join-aware uniform

Algorithm A1 Join-aware uniform at budget B

$D_j \leftarrow T_j$ for each source table with $|T_j| \leq B$ ▷ kept whole
$h(v) \leftarrow$ a number in $[0, 2^{32})$ computed from v: the first 32 bits of the MD5 hash of v's text
for each connected part of q^*'s equality joins **do**
    $k \leftarrow$ the join key of the part's largest table, if it is identifier-like with at least 50 values; else, per table, the gold's group-by or join column, or failing that the column with the most distinct values, among columns that are not measures, dates or flags and have at least 50 values
    $t \leftarrow$ the largest threshold with $|\{r \in T_j : h(r.k) < t\}| \leq B$ for every table T_j hashed on k
    $D_j \leftarrow \{r \in T_j : h(r.k) < t\}$ for those tables
    $D_j \leftarrow$ up to B rows of T_j that join the rows kept in its partner, for the part's other tables, breadth first ▷ closure
end for
return $D = \{D_j\}$

Equal values get equal numbers in every table, in DuckDB and on Redshift alike, and the numbers are spread evenly, so keeping the values with $h(v) < t$ keeps a uniform random share of them. A hashed table's pull holds the rows with the lowest hash values, up to twice the largest budget, and the threshold is chosen from those. The rows kept do not depend on the gold's conditions, so a selective gold can return no rows on a join-aware uniform sandbox.

A.2 Lineage + traps

Algorithm A2 Lineage + traps at budget B

for each place q^* reads a table T_j **do**
  $C_j \leftarrow q^*$'s `WHERE` conjuncts on T_j, and its inner-join `ON` conjuncts if it reads T_j once
  $D_j \leftarrow$ the rows of T_j that meet every condition in C_j ▷ lineage
  if $|D_j| > B/2$ **then** ▷ thin by hashing
    $k \leftarrow$ the column joining T_j to its largest partner, else the one with the most values
    $M \leftarrow \max(2, \lceil |D_j|/(B/2) \rceil)$, one modulus for all tables joined on k
    $D_j \leftarrow \{r \in D_j : |\mathsf{hash}(r.k)| \bmod M = 0\}$; if $|D_j| > B$, $M \leftarrow 4M$, at most 3 times
  end if
end for
for each table T_j and each condition $c \in C_j$ **do**
  add the trap rows of Table A1 for c, pulled on the join keys the partner tables kept
end for
add, for kept rows, the join partners that meet the partner table's own conditions
return D ▷ every addition keeps $|D_j| \leq B$

Trap	Rows added for a condition c	Catches a wrong query that
Near miss	Rows that fail c and meet the table's other conditions: for `WHERE A AND B`, `(A IS NOT TRUE) AND B` and `A AND (B IS NOT TRUE)`	drops or loosens c
Date boundary	Rows on the day just outside each date bound	shifts a date bound by a day
Alternative value	Rows with each of the three most common other values of an `IN` list or equality	uses another value set
Letter case	Rows that match a `LIKE` pattern only in another letter case	ignores letter case
Join condition	Rows that miss one `ON` conjunct	drops a join conjunct
Missing partner	Rows without a join partner	swaps an inner and an outer join
Aggregate	Near misses with nonzero values in the columns q^* aggregates	lets an excluded row into an aggregate

Table A1 | The trap rows of lineage + traps, up to 20 rows each for near misses, date boundaries, alternative values and letter case.

Lineage + traps and QAShrink select rows using rules derived from the queries they are given. Neither searches for additional wrong queries that match the gold on the sandbox but differ on W. Proposer + critic searches for these failures and seeks rows that expose them.

For lineage + traps, reducing the selected rows to fit the budget can also remove useful evidence. Hash thinning can change sums or counts, or discard rows that distinguish a wrong query from the gold. When thinning uses a column other than the join key, it can retain a row while discarding its matching row in another table, changing the join result.

A.3 Proposer + critic

Algorithm A3 Proposer + critic at budget B

$D \leftarrow$ the lineage + traps sandbox at B ▷ rows are only added, $|D_j| \leq B$
proposer session on D with the tools of Table A2 (Claude Sonnet 4.6, ≤ 60 tool calls)
for round $1, 2, 3$ **do**
    $Q \leftarrow$ five candidate mistakes the critic (Claude Opus 4.6) writes from q^*, D's row counts and rows
    $Q \leftarrow$ the queries in Q that return the gold's output on D
    drop the queries a judge (Claude Sonnet 4.6), shown q^* and the schema but not D, finds implausible
    $Q \leftarrow$ the queries in Q that are wrong on W ▷ confirmed false accepts
    if $Q = \varnothing$ **then stop**
    end if
    for each $q \in Q$ **do**
        trace an output row where q and q^* differ on W to its source rows; add them with their join partners if q then fails on D
    end for
    if some $q \in Q$ still returns the gold's output on D **then**
        proposer session from D on those queries (≤ 30 tool calls)
    end if
end for
return D

Kind	Tool	What it does
Inspect and test	`look`	Show up to 20 rows of a table of W that meet a predicate.
	`check`	Report D's rows per table against the budget, the gold's output on D and on W, and the near-miss rows D holds for each gold condition.
	`run_query`	Run a query on D and say whether it returns the gold's output.
Select rows	`add_rows`	Copy rows of a table of W that meet a predicate.
	`add_near_misses`	Copy rows that break exactly one gold condition and meet the table's others.
	`add_rows_where_differs`	Copy rows on which a gold condition and an alternative to it disagree.
Expose a wrong query	`add_rows_that_separate`	For a wrong query that returns the gold's output on D, trace an output row where the two differ on W to its source rows and copy them.
Complete joins	`fetch_partners`	Copy every row of a partner table that joins rows of D on the gold's join keys.
Edit and submit	`remove_rows`	Delete rows the proposer added that meet a predicate.
	`finish`	Submit D, refused while the gold returns no rows on D.

Table A2 | The proposer's tools. Every access to W is one bounded pull of one table, and rows added from W bring their join partners. The proposer is asked to cover mistakes other than a dropped condition: a similar column, another value set or comparison, an added filter, other `NULL` handling, another aggregate, grain or join.

The critic's candidate mistakes are not added to the held-out attempts. The critic is the model that wrote the LLM-written attempts, with the same prompt but a block describing the sandbox in place of real wrong answers; of the 36 distinct candidate mistakes confirmed wrong on W, 7 have the same text as a held-out attempt.

Hypothesis. Lineage + traps targets mistakes that drop or loosen one condition of q^*, but a wrong attempt can keep every condition and differ in a column, an aggregate or a join, and thinning can remove the rows that would expose it. A search for wrong queries that the sandbox accepts, confirmed on W, should find such gaps. The experiment compares proposer + critic as a whole with lineage + traps and does not isolate the search, since the proposer also adds more rows of the kinds lineage + traps adds than the caps of lineage + traps allow.

Table A3 separates tasks according to whether the gold query itself returns the same output on the lineage + traps sandbox and on W. Most of the gain in verdict agreement comes from tasks where these outputs differ. This grouping describes where the improvement occurs; it does not test whether proposer + critic restores the gold's warehouse output. A sandbox can preserve verdicts without reproducing that output.

Budget (rows/table)	Gold output on lineage + traps	Tasks	Lineage + traps agreement with W	Gain in agreement proposer + critic minus lineage + traps	Share of total gain
20k	Same as on W	11	0.959	+0.002	1%
	Differs from W	34	0.898	+0.039	72%
	Too big to compare	10	0.936	+0.049	27%
200k	Same as on W	14	0.969	+0.010	9%
	Differs from W	31	0.911	+0.026	52%
	Too big to compare	10	0.901	+0.059	39%
2M	Same as on W	15	0.979	+0.002	2%
	Differs from W	30	0.916	+0.030	63%
	Too big to compare	10	0.878	+0.051	35%

Table A3 | Where proposer + critic improves verdict agreement over lineage + traps. Groups compare the gold query's own output on lineage + traps with its output on W. Agreement and gain are averaged over tasks within each group. Positive gains favor proposer + critic: +0.039 means its verdict agreement is 3.9 percentage points higher than lineage + traps. Share of total gain is the group's sum of per-task gains divided by the sum over all tasks. Too big to compare means either gold output was too large to store.

B Further budget curves

Checks on proposer + critic's gain. On three tasks, the gold returns no rows on a lineage + traps sandbox but does on W: on one at every budget, and on one each at 20k and at 2M, where it does on proposer + critic's sandbox. Without them, the gain in verdict agreement is 0.031 at 20k ($p < 0.001$) and 0.015 at 2M ($p = 0.008$). A quarter of the LLM-written attempts (90 of 358 at 20k) have the same text as a query the proposer ran or the critic found wrong on W while building the sandbox (the critic's other queries are not stored); on the other attempts, the gain at 20k is 0.025 ($p = 0.010$; 9 tasks up, 2 down). Most of the gain comes from the proposer before any critic round (Table B1).

	Verdict agreement			Wrong attempts sharing a row with the gold, rejected		
Gain over lineage + traps	20k	200k	2M	20k	200k	2M
Proposer, before the critic rounds	0.026	0.023	0.020	0.073	0.059	0.040
Critic rounds, on top of it	0.008	0.004	0.006	0.035	0.023	0.019
Proposer + critic	0.033	0.028	0.026	0.108	0.082	0.059

Table B1 | Proposer + critic's gain over lineage + traps, split between the proposer's session and the critic rounds.

Controls built from the gold's rows. Lineage only is obtained by filtering the lineage + traps sandbox using the gold's per-table conditions and joins. Rows that fail the conditions on their table are removed, but supporting rows introduced during trap construction can remain. On every task and budget, the gold returns the same output on lineage only as on lineage + traps.

Lineage + join-aware uniform fills lineage only, and lineage + traps + join-aware uniform fills lineage + traps, with rows of the join-aware uniform sandbox at the same budget, by whole key values, until a table holds B rows or the join-aware uniform sandbox has no more; on the 23 tasks where join-aware uniform falls back to uniform rows, and on up to 5 tasks where the key fill failed, they add rows one at a time in a fixed hash order. The task whose gold reads the current date is left out, as are the attempts whose outputs, rescored on lineage + traps or join-aware uniform, differ from the stored ones, which leaves 54 tasks and 1,060 attempts (Table B2).

Lineage + traps has 0.024 higher verdict agreement than lineage only at 20k ($p = 0.003$), mostly by rejecting wrong attempts that share a row with the gold (0.104, $p < 0.001$), and the join-aware uniform rows raise it by 0.018 ($p = 0.017$); we detect no difference in verdict agreement between lineage + join-aware uniform and lineage + traps at any budget (0.006 less at 20k, $p = 0.42$; 0.001 less at 200k, $p = 0.90$; 0.021 more at 2M, $p = 0.26$). The two reject different wrong attempts: of the 106 that lineage only accepts at 20k, lineage + traps rejects 27 and lineage + join-aware uniform 24, 15 of them both. Lineage + traps + join-aware uniform rejects all 36, and at 2M all 62 of 91, so its verdict agreement is higher than that of lineage + traps (by 0.007 at 20k, 0.002–0.016, $p = 0.043$; 0.009 at 200k, $p = 0.031$; 0.032 at 2M, 0.011–0.083, $p = 0.016$) and of lineage + join-aware uniform (by 0.013, 0.010 and 0.011; $p = 0.004$, 0.016 and 0.002). Its row-similarity agreement on wrong attempts that share a row with the gold is also higher than that of lineage + traps at 200k (by 0.019, $p = 0.004$) and 2M (by 0.030, $p = 0.006$). It rejects two correct attempts at 20k and 200k and one at 2M that lineage + traps accepts, all of which lineage + join-aware uniform rejects too, and accepts one at each budget that lineage + traps rejects. Proposer + critic still agrees more often than lineage + traps + join-aware uniform at 20k (by 0.025, $p = 0.012$), with no detected difference at 2M ($p = 0.66$).

	Verdict agreement			Wrong attempts sharing a row with the gold, rejected		
	20k	200k	2M	20k	200k	2M
Join-aware uniform	0.655	0.718	0.781	0.421	0.511	0.596
Lineage only	0.893	0.898	0.892	0.659	0.685	0.691
Lineage + join-aware uniform	0.911	0.923	0.946	0.749	0.786	0.831
Lineage + traps	0.917	0.923	0.925	0.763	0.794	0.832
Lineage + traps + join-aware uniform	0.924	0.933	0.957	0.787	0.811	0.867
Proposer + critic	0.950	0.950	0.950	0.861	0.865	0.880

Table B2 | Verdict agreement on 1,060 attempts (54 tasks), and the share of the 293 wrong attempts that share a row with the gold on the warehouse (49 tasks) that a sandbox rejects, for the controls built from the gold's rows beside the paper's three methods.

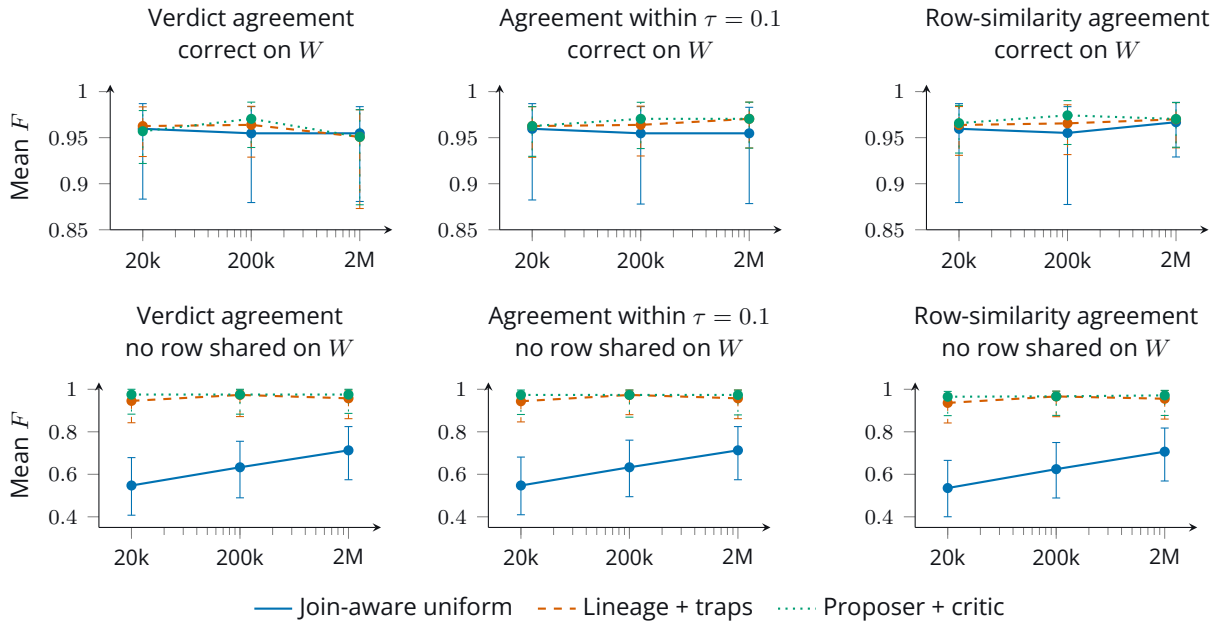


Figure B1 | Fidelity on the attempts Figure 1 leaves out. Top: correct on the warehouse. Bottom: wrong on the warehouse and sharing no row with the gold; on join-aware uniform many return no rows, as the gold does.

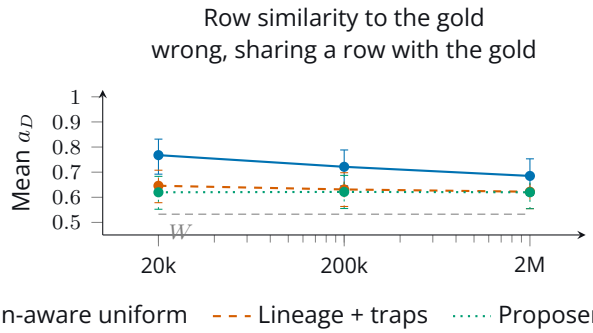


Figure B2 | Row similarity to the gold on the sandbox (a_D) for the attempts of Figure 1; dashed, on the warehouse (a_W). Above the line, the sandbox rates attempts closer to the gold than the warehouse does.

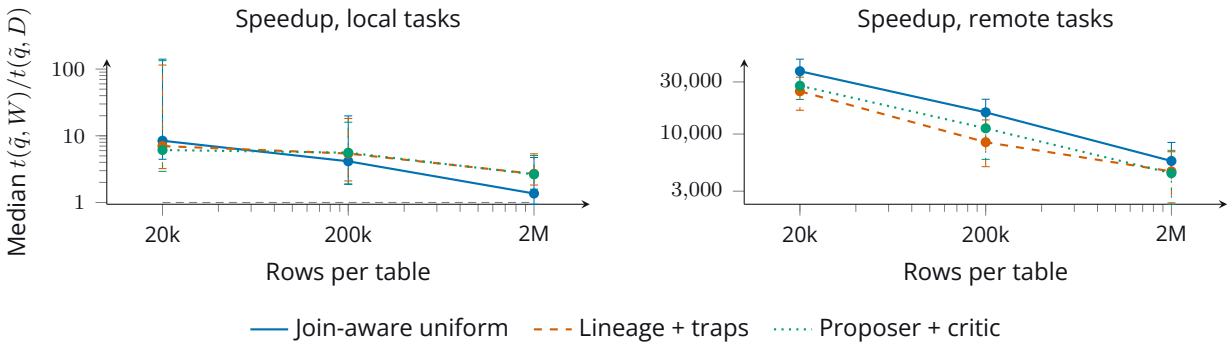


Figure B3 | Median speedup per attempt: time on W divided by time on the sandbox. Left: local tasks, W in DuckDB on the sandbox's host. Right: remote tasks, W on Redshift.