Deterministic vs. LLM-Based Validation for SQL Column-Level Lineage

Thilo Wilts, Feb. 2026.

Executive Summary

This report analyzes an experiment comparing two validation architectures (Critics) in an LLM-assisted SQL lineage extraction pipeline: a deterministic rule-based Critic versus an LLM-based Critic. The core conclusion is that deterministic validation greatly outperforms LLM-based validation for compliance-sensitive lineage extraction. In the experiment, the deterministic Critic achieved Precision 0.8812 and Recall 0.9889 (TP/FP/FN = 89/12/1), whereas the LLM Critic had similar Precision (0.8701) but much lower Recall (0.7444) (TP/FP/FN = 67/10/23). Notably, the LLM Critic produced empty lineage outputs in 20% of the test cases (no edges accepted) and encountered format failures in others. The LLM-based run was also about 2.47× slower overall.

These results imply that, for compliance-driven scenarios, a deterministic Critic provides far more consistent and auditable outcomes. The probabilistic LLM Critic leads to unpredictable gaps ("harmless but useless" refusals) and reduced coverage, which is unacceptable for audit and compliance purposes.

Introduction

SQL column-level lineage (fine-grained data provenance) describes exactly which source table columns contribute to each output column of a query. This lineage information underpins impact analysis, root-cause data quality investigations, auditing, and regulatory compliance. Provenance frameworks (e.g. W3C PROV) emphasize that lineage must be correct and explainable to ensure trust. In data governance programs, accurate lineage supports accountability and compliance processes.

Modern systems are exploring LLMs to perform lineage extraction because LLMs can parse complex SQL logic and schemas. However, LLMs are known to hallucinate – producing plausible but incorrect information. Recent studies on using LLMs as evaluators also show that they can introduce biases and instability when judging other model outputs [12]. In other words, an LLM-based Critic might reject or alter proposed lineage edges unpredictably. This experiment investigates whether such an LLM Critic can reliably enforce lineage correctness, or whether a deterministic, rule-based Critic is needed to meet compliance requirements.

Experimental Setup and Evaluation Methodology

The pipeline under test is two-stage:

- **Detective (Generator):** A large language model (LLM) that takes a SQL query and schema (table/column definitions) as input and outputs a set of proposed lineage edges in JSON format. Each edge is of the form *sourceTable.column* → *targetTable.column*.
- **Critic (Validator):** Validates and possibly filters the Detective's edges before finalizing them. Two Critic designs were compared:
- **Deterministic Critic:** A rule-based module that checks each proposed edge against schema and SQL semantics. It verifies table/column existence, CTE and alias scope, join predicates, and plausibility constraints. Implausible or invalid edges are rejected or flagged. This Critic is non-probabilistic and its logic is fully specified.
- **LLM-Based Critic:** A second LLM instance that receives the SQL, schema, and the Detective's output. It is prompted to label each edge as valid or invalid (and optionally fix obvious issues). The result is parsed as JSON for accepted edges. This Critic is heuristic and may produce different outputs across runs.

All other conditions were identical between the two runs: the same 20-case SQL test corpus (synthetic but covering realistic and adversarial patterns), the same GPT model version (`gpt-oss-20b-cloud`), execution environment, and logging. The maximum allowed reflection loops was 3 for the LLM Critic (it could attempt to refine its output up to 3 times).

SQL Test Corpus: The 20 queries were designed to stress lineage extraction. They include deeply nested CTEs, self-joins with repeated aliases, window functions, UNION-based aggregations, and intentionally tricky cases: references to unknown tables/columns, typographical schema errors, missing join conditions, ambiguous unqualified columns, and illicit cross-CTE leaks. These patterns target known failure modes in lineage tools [4].

Evaluation Metrics: We evaluate the *post-critic* output edges against ground truth. Metrics are computed at the edge level: - **Precision:** fraction of accepted edges that are correct (TP/(TP+FP)). - **Recall (coverage):** fraction of true edges that were accepted (TP/(TP+FN)). - **F1 Score.**

We emphasize precision as primary (to prevent false lineage claims) but still report recall because missing edges can undermine downstream processes. We also log the number of false positives (invalid edges accepted) vs. false negatives (valid edges missed). Additional metrics include: - **Empty-Lineage Cases:** Count of cases where the Critic produced zero edges. - **Format/Schema Errors:** Cases where the Critic output was invalid JSON or incomplete. - **Performance:** Total and per-case runtime, and number of loops used by the LLM Critic.

This setup isolates the Critic as the *only variable*. Both runs saw identical Detective outputs before validation. Thus differences in final output are attributable to the validation strategy alone.

Results

Aggregate Performance: The overall lineage accuracy is shown in Table 1. The deterministic Critic achieves **Precision 0.8812** and **Recall 0.9889**, whereas the LLM Critic achieves **Precision 0.8701** and **Recall 0.7444**. In absolute terms, the LLM Critic missed 23 true edges (false negatives) versus only 1 for the deterministic Critic. The precision gap is modest, meaning both largely avoided hallucinations. The big difference is coverage.

Table 1:Overall lineage accuracy.

Critic Type	Precision	Recall	F1	TP	FP	FN
Deterministic	0.8812	0.9889	0.9319	89	12	1
LLM-based	0.8701	0.7444	0.8024	67	10	23

Pattern Breakdown: Table 2 examines performance by query category. For clean/noise-free queries, both critics perform well, but the LLM Critic's recall suffers even there. For example, on the *none* (clean) subset, the deterministic Critic has Precision 0.9012 and Recall 0.9865, while the LLM Critic has 0.8667 precision but only 0.7027 recall (missing 22 of 73 true edges). Notably, when the SQL contained a simple typo or missing schema element, the LLM Critic chose to reject *all* offending edges (achieving 1.0000 precision) at the cost of recall. Conversely, both critics maintained high recall on queries with missing join predicates.

Splitting by difficulty reveals a **non-monotonic brittleness**: on *hard* queries, the LLM Critic attained 0.8571 recall, but on the *medium* group recall fell to 0.6935, lower than on the easier ones. In contrast, the deterministic Critic remained near 100% recall across difficulties. This irregular failure pattern means that the LLM Critic's coverage cannot be reliably predicted from query complexity alone. |

Table 2: Performance by query category.

Noise Type	**Precision (Det)**	**Recall (Det)**	**Precision (LLM)**	**Recall (LLM)**
none (clean)	0.9012	0.9865	0.8667	0.7027
typo	0.7000	1.0000	1.0000	0.8571
missing_join	0.9000	1.0000	0.8182	1.0000
hard	0.8235	1.0000	0.7742	0.8571
medium	0.9104	0.9839	0.9348	0.6935

Precision vs. Recall: The deterministic Critic approaches full coverage with a recall of 98.89%, whereas the LLM Critic's coverage is significantly lower at 74.44%. This gap in recall exists despite the two methods maintaining very similar precision, with the deterministic model at 88.12% and the LLM-based model at 87.01%.

Catastrophic Failures: In 4 out of 20 queries (20%), the LLM Critic produced *zero* output edges (i.e. a completely empty lineage). Table 3 lists these cases. Two failures were *format errors* (the LLM output was not valid JSON after max loops), and the other two were silent

refusals. For example, in `deep_cte_sales_waterfall`, the LLM Critic returned "SUCCESS" but no edges (all 4 true edges were dropped). The deterministic Critic, in contrast, produced partial lineage for these queries. The loss of *all* true edges in these cases (FN=4–6 each) explains almost all of the LLM Critic's low recall.

Table 3: Summary of test cases.

Test Case	Outcome	Loops Used	Runtime (s)	TP/FP/FN	Notes
deep_cte_sales_waterfall	SUCCESS	2	52.47	0/0/4	Silent failure: no edges despite "SUCCESS" status
schema_violation_ unknown_columns	FAIL_FORMAT	3	90.20	0/0/5	Format error after max loops
wrong_but_schema_ valid_join_key	PARTIAL	2	52.32	0/0/6	All edges pruned (missing join-key)
cte_passthrough_ chain_cast	FAIL_FORMAT	3	73.46	0/0/6	Format error after max loops

Runtime & Loops: The LLM Critic run was significantly slower. Total end-to-end time was ~968 seconds vs. ~392 seconds for the deterministic run (~**2.47× slower**). The average loops used per case was 1.7 for the LLM Critic (max allowed 3) vs 1.2 for deterministic. Notably, all FAIL_FORMAT cases hit the 3-loop limit. The extra LLM calls and retries did not improve coverage; they only increased latency and, in two instances, still failed to produce valid output.

Failure-Mode Analysis

The differences in failure modes between the critics have direct governance implications:

- **"Harmless but Useless" Anti-Pattern:** The LLM Critic often avoids errors by refusing edges altogether. This yields high precision but sometimes **zero coverage**. This is similar to an AI that always says "I don't know" – safe in content but useless in function [15]. In our context, an empty lineage is of little practical value: it fails to inform impact analysis or auditing. From a risk standpoint, a lineage with no edges is as bad as a false-positive lineage, because it provides no actionable trace and conceals true dependencies.

- **Empty-Lineage False Negatives:** Missing an edge (a false negative) means an actual data dependency is not recorded. In a compliance environment, that gap could hide a critical data flow. Because the LLM Critic sometimes discarded entire sets of edges, it committed many such false negatives. Even a single critical false negative can violate data governance rules (e.g., undisclosed personally identifiable

data flows). The deterministic Critic missed virtually none, highlighting how a consistent rule-based check is essential for completeness.

- **Format Instability:** The LLM Critic itself suffered parsing failures. Two cases ended in *FAIL_FORMAT* after three loops. This means the model, when prompted to "output valid JSON", failed to comply. Such syntactic failures represent an additional error surface introduced by the LLM. A deterministic Critic's output is inherently structured by design, so it cannot produce JSON errors. In production, format errors from validation could break downstream systems or raise false alarms.

- **Schema Reasoning Errors:** In cases like `wrong_but_schema_valid_join_key`, the LLM Critic chose to reject all edges when the join key appeared semantically invalid, even though a valid lineage existed. This brittleness in schema reasoning – not recognizing a valid dependency – contrasts with the deterministic Critic which at least returned partial lineage. It suggests the LLM Critic's checks were overly strict or misinterpreted SQL logic.

- **Non-Monotonic Brittleness:** The performance on *medium* queries was worse for the LLM Critic than on *hard* ones. This unpredictability is itself a failure mode: if more complex cases sometimes succeed but simpler ones fail, engineering confidence cannot be maintained. This aligns with research showing LLM-based evaluations can vary erratically with prompt framing and input structure [11,12].

Governance & Production Recommendations

Based on these findings, the following practices are recommended for production lineage systems:

- **Use a Deterministic Gatekeeper:** Make the deterministic Critic the final authority. Its rule-based checks yield reproducible decisions and clear audit trails, which satisfy governance requirements for transparency. Even if an LLM produces candidate edits or flags, a deterministic validation layer should confirm or reject them.

- **LLM Critic as Advisory Only:** If an LLM Critic is used, treat it as a *suggestion engine*, not a decisive filter. For example, let it highlight suspicious edges or potential fixes, but ensure a rule-based system enforces final correctness. This hybrid approach leverages the LLM's flexibility without relinquishing control.

- **Enforce Non-Empty-Lineage Policies:** Implement a policy that lineage outputs cannot be empty unless the query is provably trivial (e.g. no source columns). An empty result should trigger an exception or human review. For compliance alignment: document each lineage run, and if coverage falls below a threshold, log an error (the EU AI Act, for instance, requires detailed logging and technical documentation for high-risk systems).

- **Loop and Failure Circuit-Breakers:** Set clear limits on iterative refinement. If the LLM Critic fails to produce valid output after N loops (N=3 was used here), abort and fallback to the last valid state or to a rule-based resolution. Since multiple loops increased latency but did not recover missing edges, it is safer to bail out early and flag the case.

- **Schema and Format Guards:** Before trusting the Critic's output, verify JSON/schema integrity with deterministic checks. Any format errors should be caught immediately. Similarly, enforce that each accepted edge must match an existing column reference, preventing acceptance of hallucinated columns.

- **Comprehensive Test Suite:** Regularly test the system with known challenging queries (like those in this experiment). Each time the LLM Critic exhibits a new failure pattern, either refine the rules to catch it or adjust the LLM's prompt. Continuous regression testing will help ensure no stealth failures slip into production.

These measures align with industry best practices: frameworks like NIST's AI Risk Management Framework emphasize governance controls, logging, and documentation. Data governance methodologies (e.g. DAMA DMBOK) likewise treat lineage as a first-class managed asset, requiring policies around its completeness and correctness.

Interpretation and Performance Analysis

The experimental differences reflect a fundamental trade-off between safety and usefulness. The deterministic Critic acts as a conservative gatekeeper but still preserves nearly all valid lineage. The LLM Critic, in contrast, "plays it safe" by discarding uncertain edges, which minimizes hallucinations but creates "blind spots." This is reminiscent of the safety–usefulness dichotomy discussed in AI alignment: a system optimized solely to avoid harm can end up perpetually deferring or refusing tasks [15].

From a technical perspective, the LLM Critic's failures are not surprising. LLMs lack an inherent notion of SQL schema consistency and must infer it from context. When faced with ambiguity or complex scope (e.g. multiple CTEs), the model's probability estimates may default to the "no edge" answer for safety. However, this violates the intended function of a lineage tool. In mission-critical pipelines, missing a lineage edge (false negative) is often worse than a correct edge that needs flagging, because the latter at least carries information.

The non-monotonic brittleness is particularly concerning. It shows that query complexity is not a smooth predictor of success; even moderate queries stymied the LLM Critic unpredictably. For engineering, this means that test coverage must include a wide variety of patterns, and one cannot assume "coverage will improve on simpler inputs." The deterministic logic, by contrast, has a monotonic characteristic: adding more rules or cases would systematically improve coverage.

On performance, the LLM Critic's 2.47× slowdown and higher loop count imply a significant cost in real deployments. If the pipeline were time-sensitive, this overhead might be unacceptable. More importantly, the additional loops did not yield better lineage – they just chased failures. In production, one would allocate more computational and latency budget for an LLM step that only occasionally helps. Thus, from a cost-benefit standpoint, the LLM Critic introduced overhead without delivering recall gains.

From a governance standpoint, these results reinforce best practices in regulated AI: use probabilistic models with strong deterministic checks and traceability. The NIST AI RMF and related guidelines advise combining automated AI components with manual or rule-based safeguards. In this case, we see that deterministic oversight is necessary to meet auditability and completeness requirements. Similarly, the EU AI Act emphasizes accountability and record-keeping: lineage outputs and any decisions to reject lineage edges should be logged and explainable.

Limitations

This evaluation has limitations. Only one run of each Critic was performed, and some randomness remains in the LLM outputs. The SQL corpus is synthetic and relatively small, though carefully designed. Real-world data warehouses may include additional SQL dialect features or complexities not captured here. Also, the focus was on precision; in contexts where completeness is equally mandated, the recall gaps observed would be even more critical. Future work should include multiple trial runs, alternative LLM models/prompts, and perhaps integration with formal provenance engines as additional baselines.

References

[1] J. Cheney, L. Chiticariu, and W.-C. Tan, *"Provenance in Databases: Why, How, and Where," Foundations and Trends in Databases*, vol. 1, no. 4, pp. 379–474, 2009.

[2] P. Buneman, S. Khanna, and W.-C. Tan, *"Why and Where: A Characterization of Data Provenance,"* in *Proc. ICDT*, 2001, pp. 316–330.

[3] T. J. Green, G. Karvounarakis, and V. Tannen, *"Provenance Semirings,"* in *Proc. PODS*, 2007, pp. 31–40.

[4] T. Müller, B. Dietrich, and T. Grust, *"You Say 'What', I Hear 'Where' and 'Why': (Mis-)Interpreting SQL to Derive Fine-Grained Provenance,"* in *Proc. VLDB*, 2018.

[5] World Wide Web Consortium, *PROV-DM: The PROV Data Model – Primer*, Apr. 2013.

[6] V. Khatri and C. V. Brown, *"Designing Data Governance,"* Commun. ACM, vol. 53, no. 1, pp. 148–152, 2010.

[7] DAMA International, *DAMA-DMBOK: Data Management Body of Knowledge*, 2nd ed., Technics Publications, 2017.

[8] National Institute of Standards and Technology, *"Artificial Intelligence Risk Management Framework (AI RMF 1.0),"* NIST AI 100-1, 2023.

[9] European Union, *Regulation (EU) 2024/1689 on Artificial Intelligence (AI Act)*, Official Journal L 305, July 2024.

[10] Z. Ji et al., *"Survey of Hallucination in Natural Language Generation,"* ACM Computing Surveys, vol. 56, no. 1, 2023.

[11] L. Zheng et al., *"Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena,"* arXiv:2306.05685, 2023.

[12] L. Shi et al., *"Judging the Judges: A Systematic Study of Position Bias in LLM-as-a-Judge,"* arXiv:2406.07791, 2024.

[13] A. Madaan et al., *"Self-Refine: Iterative Refinement with Self-Feedback,"* arXiv:2303.17651, 2023.

[14] Y. Liu et al., *"G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment,"* in *Proc. EMNLP*, 2023, pp. 2511–2522.

[15] Y. Bai et al., *"Constitutional AI: Harmlessness from AI Feedback,"* arXiv:2212.08073, 2022.