A Skill-Packaged Evaluator for Production Text2SQL Agents

Jiaxiang Ren Aaditya Shukla Meghana Puvvadi Min Du Anbang Xu Rama Akkiraju
NVIDIA

Abstract

Text2SQL evaluation is often reduced to benchmark scoring over generated SQL, but production database agents also route requests, select execution surfaces, condition on runtime context, execute queries, and expose diagnostics. We present a Text2SQL Evaluation Skill that packages this workflow as reusable procedural knowledge. Given natural-language questions, reference SQL, expected route metadata, and database access, the Skill runs an agent or consumes predictions, executes reference and predicted queries, and reports routing, SQL structural, execution, and diagnostic evidence. An anonymized production case study and a portable SQLite demo show that structural and execution signals can materially disagree, motivating multi-signal evaluation with explicit adapter, comparison, and safety policies. The `SKILL.md` with codes is available at https://github.com/reckdk/text2sql-evaluator-skill.

CCS Concepts: • **Computing methodologies** → **Natural language processing**; • **Information systems** → *Query languages.*

Keywords: Text2SQL, database agents, agent skills, evaluation, execution accuracy, safety

ACM Reference Format:
Jiaxiang Ren Aaditya Shukla Meghana Puvvadi Min Du Anbang Xu Rama Akkiraju. 2026. A Skill-Packaged Evaluator for Production Text2SQL Agents. In *Agent Skills '26: The First Workshop on Agent Skills, May 26, 2026, San Jose, CA.* ACM, New York, NY, USA, 7 pages.

1 Introduction

Text2SQL systems promise natural-language access to structured data, but the task has moved well beyond translating an utterance into a single SQL string. Classic benchmarks such as Spider established cross-domain generalization as a central challenge for semantic parsing [11]. More recent benchmarks make the deployment setting harder and more realistic: BIRD emphasizes large databases, noisy or incomplete database values, external knowledge, and query efficiency [4], while Spider 2.0 frames Text2SQL as enterprise workflow evaluation over heterogeneous systems, database metadata, dialect documentation, and longer multi-step SQL workflows [3]. In this setting, a deployed Text2SQL system is closer to an agentic database workflow than to a standalone parser. It may select a data source, retrieve schema context, planning, generate SQL or another query language, execute against a backend, and return an answer-bearing table.

This shift exposes a gap in evaluation. Exact match is brittle because many SQL strings can express the same intent; structural match is useful but cannot guarantee predicates, joins, aggregation, or temporal scope; and execution accuracy remains snapshot-dependent. Test-suite evaluation improves single-snapshot execution by comparing over multiple distilled databases [12], but recent work still highlights ambiguity, gold-query defects, and match-function bias [7]. Verification-oriented analyses show that queries can coincide on one database while differing elsewhere [2]. Benchmark scores are therefore valuable but insufficient as a reusable, safety-aware procedure for production Text2SQL agents.

Production agents add failure surfaces before and after SQL generation. A query may be structurally plausible but routed to the wrong source, generated for the wrong dialect, grounded in an incomplete schema subset, executed against a moving snapshot, or unsafe to run. Recent multi-stage systems, modular benchmarks, and execution-guided training studies already reflect this reality through schema selection, retrieval, refinement, validation, and execution feedback [1, 5, 8–10]. FusionSQL addresses a complementary setting: estimating performance on unseen, unlabeled workloads [6]. Our setting assumes labeled examples, reference SQL, expected routes, or expected behavior are available, and asks how to evaluate the actual agent workflow with diagnostics rather than only an aggregate pass rate.

We present a Text2SQL Evaluation Skill: a reusable procedural artifact for evaluating database agents end to end. In the Agent Skills framing, the artifact includes a Skill spec, configuration schema, adapter, demo dataset, runner, and report schema. It returns routing, SQL structural, and execution signals rather than claiming full semantic equivalence. Our contributions are: (1) a Skill formulation that runs or consumes an agent workflow, executes reference and predicted queries, and reports diagnostic evidence; (2) a multi-signal metric contract with policies for aliases, column and row order, snapshots, and unsafe-query rejection; (3) validation through anonymized production runs and a portable SQLite demo; and (4) compatible with popular harnesses like Claude Code and Codex. Figure 1 summarizes the workflow.

2 Text2SQL Evaluation Skill Design

The Skill evaluates an agentic database workflow through three partial, policy-relative signals rather than a single

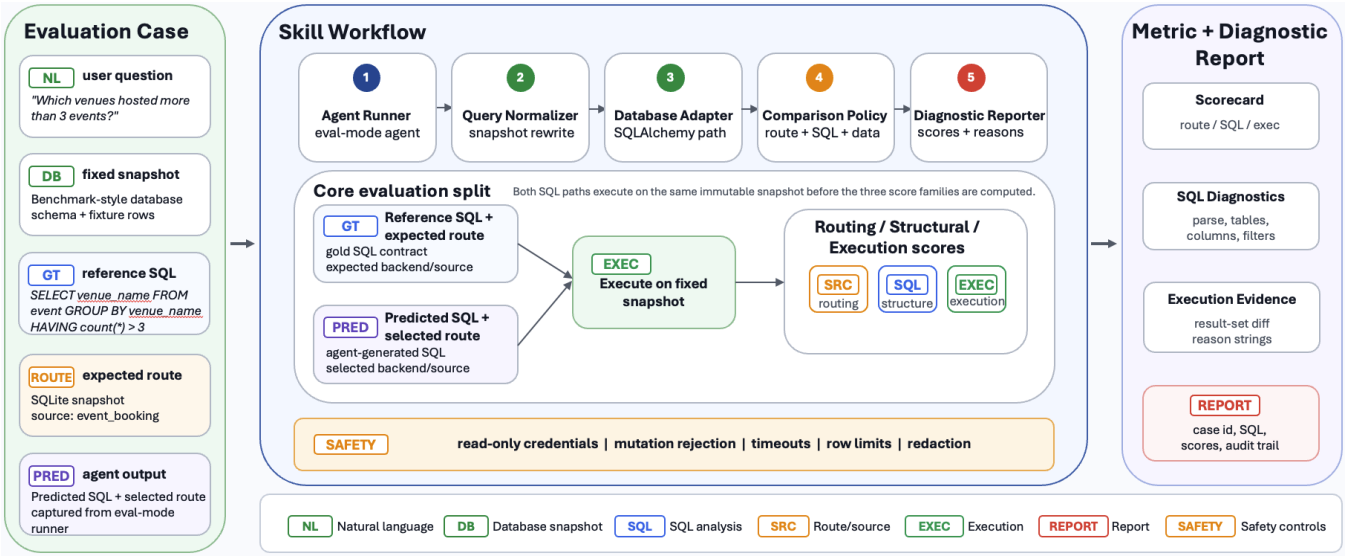

Figure 1. Overview of the Text2SQL Evaluation Skill workflow. The Skill packages an evaluation case, an agentic workflow, adapter-based execution on a fixed snapshot, policy-aware comparison, and a metric-plus-diagnostic report with routing, structural, and execution evidence.

scalar. Routing correctness measures whether the agent selected the intended execution surface; structural correctness checks whether the predicted query uses expected data surfaces and answer-bearing expressions; and execution agreement measures denotational agreement on a fixed snapshot under a comparison policy.

2.1 Problem Formulation

Let the evaluation set be

$$\mathcal{D} = \{(q_i, y_i^\star, d_i, c_i, B_i)\}_{i=1}^{n},$$

where q_i is a request, $y_i^\star$ a reference query, d_i the expected route, c_i schema/runtime context, and B_i the database snapshot. A route may denote a SQL engine, semantic layer, or tool-specific query surface.

Given the request and context, the evaluated agent returns

$$A(q_i, c_i) \to (\hat{y}_i, \hat{d}_i, m_i),$$

where $\hat{y}_i$ is the predicted query, $\hat{d}_i$ the selected route, and m_i metadata such as planning traces, tool calls, retrieved schema, or execution logs. The Skill executes reference and predicted queries under configured safety and comparison policies:

$$R_i^\star = \mathrm{Exec}(d_i, y_i^\star; B_i), \qquad \hat{R}_i = \mathrm{Exec}(\hat{d}_i, \hat{y}_i; B_i).$$

Execution may return a relation, fail, time out, or be blocked by the unsafe-query guard; the report preserves these outcomes.

The evaluator reports a score vector

$$\mathbf{s}_i = (s_i^{\text{route}}, s_i^{\text{struct}}, s_i^{\text{exec}}).$$

The routing score checks selected route, the structural score checks query-form evidence, and the execution score compares returned relations. These are diagnostic proxies, not proofs of SQL semantic equivalence.

Run-level results are mean scores over configured examples. Missing metrics are unavailable; execution errors, timeouts, and blocked mutation queries count as execution failures unless explicitly out of scope. The report also records structural/execution disagreement when query-form evidence and answer-result evidence diverge.

2.2 Skill Interface and Modules

The Skill has four modules. The **agent runner** invokes a production workflow or consumes supplied predictions and normalizes outputs into SQL, selected route, and metadata. The **database adapter** exposes a common read-only execution interface; the runnable testing artifact uses a SQLAlchemy-compatible SQLite path, while production deployments can bind enterprise backends. The **metric evaluators** compute routing, structural, and execution scores while preserving row-level evidence; the portable artifact uses `sqlglot` AST parsing for SQL structure and a pure-Python relation comparator for execution result checks. The **diagnostic reporter** aggregates failures by likely cause, including wrong route, table mismatch, missing filter, aggregation mismatch, execution failure, and unsafe-query block. This interface makes the evaluator a Skill rather than a one-off script: `SKILL.md` is the task-local evaluation contract, host prompt management can scope Skill loading, and implementations can replace runners or adapters while preserving the metric contract and report format.

2.3 Metric Family

The main metric contribution is the decomposition of agentic Text2SQL evaluation into complementary signals. Routing isolates planner/tool-selection behavior: in a multi-source workflow, plausible SQL can fail because it targets the wrong backend.

The structural score compares observable SQL evidence, mainly physical table references and selected answer expressions. It relaxes alias names and projection order, and does not treat extra predicted columns as fatal when expected answer columns are present. This is a diagnostic policy, not a definition of SQL meaning.

One operational instance computes structural score from a table-match indicator τ_i and selected-expression recall ρ_i:

$$s_i^{\text{struct}} = \begin{cases} \alpha + (1-\alpha)\rho_i, & \tau_i = 1, \\ \beta\rho_i, & \tau_i = 0. \end{cases}$$

Here τ_i records table match and ρ_i selected-expression recall. In our production configuration, $\alpha = 0.3$ and $\beta = 0.2$. These weights are operational triage parameters, not a universal Text2SQL metric. The scalar is used for triage; table match, expression recall, and execution agreement are also reported, so conclusions need not depend on a single weighting choice.

The execution score compares returned dataframes on the same snapshot. It is the strongest observable signal, but remains policy-relative: unordered rows, duplicates, `NULL`s, floats, timestamps, collation, and top-k ordering require explicit choices. A high structural score with a low execution score often indicates a missing filter, wrong aggregation, incorrect join, or temporal grounding error. A low routing score with high structural similarity points instead to tool selection.

2.4 Robustness and Safety Policy

The robustness policy encodes production comparison choices explicitly. Alias names and projection order are relaxed when normalized expressions match; rows are order independent unless top-k or ranking semantics require order; volatile tables may require snapshot normalization. The safety policy treats generated SQL as executable authority: implementations should use read-only credentials, reject mutations, enforce timeouts and row limits, redact reports, and validate adapter dependencies. SQL execution can run in sandbox with restricted operations and policies when available. The SQLAlchemy-compatible path supports portability, but dialect behavior still requires backend validation; the contribution is a reusable evaluation Skill, not a benchmark replacement.

3 Evaluation: Production Case Study and Portable Demo

We evaluate the Skill along two axes: whether it exposes distinct failure modes in production-like agent workflows, and whether the same procedure can be shared as a runnable artifact outside the original deployment. Production validation and portable reproducibility are reported separately. The production case study supplies evidence that the metric vector is useful in a deployed workflow; the SQLite demo supplies evidence that the Skill interface, comparison policies, and report schema can be exercised locally without secrets or private infrastructure.

The anonymized enterprise case study consists of two production evaluation configurations over labeled Text2SQL examples. Both configurations run the agent workflow, extract generated SQL and metadata, execute reference and predicted queries, and compute structural and execution scores. Case study A is a filtered snapshot-oriented run of 255 examples focused on SQL and result comparison; data source routing was not evaluated as a separate logged metric in that configuration. Case study B is a broader fallback-enabled run of 343 examples that additionally records selected route, making routing score available. The two rows should therefore be read as complementary production measurements, not as a controlled ablation or leaderboard comparison.

The central observation is that structural and execution signals disagree often enough to matter. In case A, the structural score is 0.968 while execution score is 0.831, with 20% structural and execution disagreement under operational thresholds. This is exactly the failure profile that a single SQL-structure score would obscure. For example, a predicted query can select the expected table and column but omit a task-critical filter such as active snapshot date, region, or status. Under the structural metric, the table and selected expression match the reference query; under execution, the returned records include rows outside the intended slice and receives a low score. This pattern points developers toward predicate generation, temporal grounding, or query refinement rather than data source routing or schema retrieval.

Case B shows why routing should be reported separately. A multi-source database agent may generate a query that appears plausible in isolation but is directed at the wrong backend or semantic layer. In such cases, execution failure or answer mismatch is a downstream symptom of a planner or tool-selection error. The routing score of 0.890 therefore supplies a separate diagnostic channel: improving schema linking or data source selection may be more important than changing the SQL generator. Conversely, when routing is correct but execution score is low, the likely causes move to dialect details, predicates, joins, aggregation, or result comparison policy.

The portable SQLite demo is intentionally small. It contains 11 synthetic examples covering exact match, alias-only differences, column-order differences, row-order differences when order is not semantically required, wrong-table errors, missing filters, aggregation mismatch, extra

Table 1. Evaluation results. Scores are means over configured examples; "–" means the metric was not logged in that configuration.

Setting	# Ex.	Route	Struct.	Exec.	Key finding
Production case study A	255	–	0.968	0.831	Structural SQL scores can overestimate result correctness; 20% of examples show material structural/execution disagreement.
Production case study B	343	0.890	0.830	0.802	Routing, SQL structure, and execution scores expose different classes of failures.
Portable SQLite demo	11	0.909	0.805	0.682	The Skill runs locally without secrets and demonstrates alias, order, mutation rejection, and diagnostic behavior.

predicted columns, empty-result agreement, blocked mutation SQL, and data source route mismatch. Its summary metrics are route 0.909, structural 0.805, execution 0.682, structural/execution disagreement rate 0.273, and one mutation-query block. The cleanest diagnostic example is the missing-filter case: the predicted SQL receives structural score 1.0 because `sqlglot` extraction finds the expected physical table and selected expressions, but execution score 0.0 because the returned values do not match the reference result.

The demo is not an empirical benchmark or cross-database validation. It demonstrates the Skill interface and policies on SQLite, while the production runs provide deployment evidence.

For artifact inspection, the report keeps row-level evidence, including parser details, extracted SQL structure, execution shapes, comparison policy, safety flags, and representative disagreements.

4 Discussion: Robustness, Security, and Limitations

Evaluation validity. The Skill improves observability but inherits the limits of reference SQL, snapshots, and comparison policies. A single reference query may not capture every valid interpretation of an ambiguous question. Execution agreement is denotational agreement on a fixed state, not a proof of intent; structural scoring is a diagnostic proxy, not a model-independent account of SQL meaning. The metric vector makes uncertainty inspectable by separating route selection, query form, and returned result, which helps distinguish planner error, schema grounding, dialect mismatch, query generation, execution failure, and comparison policy.

Robustness policies. Alias normalization, column-order independence, row-order treatment, and snapshot normalization are evaluation assumptions rather than cosmetic details. For many tabular QA tasks, display aliases and projection order should not change correctness; for ranked outputs, top-k, or time series questions, row order may be essential. Moving production tables can make snapshot drift look like model error. The Skill should therefore expose how it handles duplicate rows, `NULL` values, floating point tolerance, timestamps, time zones, collation, and case sensitivity.

Portability. The adapter layer makes the evaluator portable while leaving backend validation explicit. SQLAlchemy is a practical path for relational engines, and the same high-level contract can bind to enterprise warehouses, semantic layers, or backend-specific executors. Portability does not imply identical dialect behavior: quoting, coercion, date functions, limits, and aggregation can affect both execution and comparison. The artifact is therefore a Skill with replaceable adapters and a stable report schema, not a claim of universal database equivalence. With supplied predictions, evaluation uses deterministic parsing and execution with no LLM-token cost; when invoking the agent runner, token cost scales roughly with examples times agent calls per example.

Safety and dependency risks. A Skill that executes generated SQL has operational authority, so safety belongs in the Skill specification. The artifact demonstrates mutation-query rejection as a smoke test, not an adversarial SQL firewall; production deployments should additionally use read-only roles, isolated snapshots, timeouts, row limits, adapter allowlists, dependency pinning, and redaction.

Limitations. The production case study is anonymized and cannot release private schemas, questions, snapshots, or infrastructure. The SQLite artifact is a mechanism demo, not a substitute for large public benchmarks. The structural weights are configurable operational defaults, not universally optimal. The current Skill focuses on SQL-like execution and dataframe comparison; chart correctness, answer faithfulness, clarification, and long multi-turn evaluation require additional evaluators.

5 Conclusion

Production Text2SQL agents need evaluation procedures that run the workflow, execute queries, compare results, and report where failures occur. The Text2SQL Evaluation Skill packages this procedure as reusable agentic knowledge with explicit metric contracts, adapter based execution, diagnostics, and safety controls. Production and SQLite evidence show why routing, structural, and execution signals should be reported together. Future works include additional evaluators, metrics, and self-correcting loops evolved from failure analysis.

References

[1] Shizheng Hou, Wenqi Pei, Nuo Chen, Quang-Trung Ta, Peng Lu, and Beng Chin Ooi. 2026. NL2SQLBench: A Modular Benchmarking Framework for LLM-Enabled NL2SQL Solutions. *Proceedings of the VLDB Endowment* 19, 5 (Jan. 2026), 1001–1015. doi:10.14778/3796195.3796211

[2] Rocky Klopfenstein, Yang He, Andrew Tremante, Yuepeng Wang, Nina Narodytska, and Haoze Wu. 2026. SpotIt: Evaluating Text-to-SQL Evaluation with Formal Verification. In *International Conference on Learning Representations*. Poster.

[3] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. *arXiv preprint arXiv:2411.07763* (2024).

[4] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems*.

[5] Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. 2026. Think2SQL: Blueprinting Reward Density and Advantage Scaling for Effective Text-to-SQL Reasoning. *Transactions on Machine Learning Research* (2026).

[6] Trinh Pham, Thanh Tam Nguyen, Viet Huynh, Hongzhi Yin, and Quoc Viet Hung Nguyen. 2026. An Efficient and Effective Evaluator for Text2SQL Models on Unseen and Unlabeled Data. *arXiv preprint arXiv:2603.07841* (2026). doi:10.48550/arXiv.2603.07841 Accepted at ICDE 2026.

[7] Cedric Renggli, Ihab F. Ilyas, and Theodoros Rekatsinas. 2025. Fundamental Challenges in Evaluating Text2SQL Solutions and Detecting Their Limitations. *arXiv preprint arXiv:2501.18197* (2025).

[8] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *arXiv preprint arXiv:2405.16755* (2024).

[9] Yihan Wang, Peiyu Liu, and Xin Yang. 2025. LinkAlign: Scalable Schema Linking for Real-World Large-Scale Multi-Database Text-to-SQL. *arXiv preprint arXiv:2503.18596* (2025). doi:10.48550/arXiv.2503.18596

[10] Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. 2025. ToolSQL: A Tool-Assisted Agent for SQL Verification and Refinement. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[11] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921. doi:10.18653/v1/D18-1425

[12] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. 396–411. doi:10.18653/v1/2020.emnlp-main.29

A Text2SQL Evaluation Skill Prompt

Table 2. Prompt-style artifact inventory. The anonymized runnable testing repository will be made available with the Skill prompt, demo config, synthetic SQLite dataset, database builder, adapter, evaluation runner, regression tests, and example reports.

Artifact	Role	Release form
`SKILL.md`	Evaluator prompt/specification	Appendix and testing repository
`configs/demo_eval.yaml`	Portable demo configuration	Testing repository
`examples/demo_dataset.json`	Synthetic evaluation cases	Testing repository
`scripts/run_demo_eval.py`	Runnable evaluator and tests	Testing repository

Prompt A.1: Text2SQL Evaluation Skill

```
---
name: text2sql-eval
description: >
  Evaluate Text2SQL and database-agent workflows end to end by
  running questions through an agent or supplied predictions,
  executing ground-truth and predicted SQL, scoring route selection,
  SQL structure, execution equivalence, and diagnostic reports
  with safety checks for generated SQL.
---

# Text2SQL Evaluation Skill

Use this Skill when evaluating a Text2SQL agent, database agent,
or agentic data workflow against natural-language questions with
ground-truth SQL.

## Workflow

1. Load the evaluation dataset and config.
2. For each row, get question, ground_truth_sql, and either
   predicted_sql or an agent runner that produces SQL.
3. Normalize the agent output to generated_sql and selected_data_source.
4. Reject unsafe generated SQL unless the evaluation is running in an
   explicitly isolated mutation sandbox.
5. Execute ground-truth and predicted SQL through the configured
   database adapter.
6. Compute routing correctness, SQL structural correctness, and
   execution correctness.
7. Save row-level reasoning and aggregate diagnostics.

## Required Inputs

- Dataset with stable id, natural-language question, ground_truth_sql,
  and optional predicted_sql.
- Database adapter config mapping data source names to connection URIs.
- Optional agent runner for generating SQL when predictions are not
  precomputed.

Recommended row fields:
- expected_data_source
- selected_data_source
- order_required
- allow_extra_predicted_columns
- scenario

## Metrics
```

```
Report three primary scores:
- Routing correctness: selected data source equals expected data source.
- SQL structural correctness: sqlglot-parsed physical table match plus
  ground-truth selected-expression coverage.
- Execution correctness: ground-truth and predicted result relations are
  equivalent under task-appropriate normalization.

Default structural score:
score = 0.3 + 0.7 * column_match if table_match
        else 0.2 * column_match

## Robust Comparison Rules

- Use sqlglot AST parsing for SQL structure; exclude CTE aliases from
  physical table matching.
- Align aliases through underlying expressions where possible, without
  giving full credit when the same alias names different expressions.
- Treat selected columns as order independent.
- Treat rows as order independent unless the example declares
  order_required.
- Allow extra predicted columns when expected columns and values are
  present.
- Compare original and processed SQL in the report.
- Use a pure-Python relation comparator in this portable artifact;
  production deployments may use heavier dataframe tooling behind the
  same metric contract.

## Safety Rules

Before executing generated SQL:
- Use read-only credentials.
- Reject INSERT, UPDATE, DELETE, MERGE, CREATE, DROP, ALTER, TRUNCATE,
  GRANT, and REVOKE by default.
- Apply row limits and query timeouts where the adapter supports them.
- Never log secrets or full connection strings in reports.
- Record execution errors separately from semantic mismatches.

## Outputs

Produce:
- workflow_output.json
- routing_eval_output.json
- sql_correctness_output.json
- dataframe_comparison_output.json
- post_evaluation_analysis_report.json

The report should include aggregate scores, per-row reasoning, and
structural/execution disagreement rate.
```