Beyond Exact Match: Evaluating Language-Model Analytics Systems from Benchmark to Production

Literature review and conceptual analysis

Abstract

Evaluation practices for language-model analytics systems are fragmented. Text-to-SQL research offers controlled benchmarks and reproducible measures, while production reports emphasize completion rates, time savings, adoption, and user experience. Neither perspective alone establishes whether an analytics agent is dependable. This evidence synthesis uses SiriusBI and SiriusDeliver as connected cases: SiriusBI reports a modular conversational BI system with multi-round clarification and alternative SQL-generation strategies; SiriusDeliver reports an agent for warehouse task delivery with hierarchical skills, artifact lifecycle control, and trace-driven evolution. The article develops an evaluation matrix spanning semantic intent, executable artifacts, workflow outcomes, human work, governance, and longitudinal effects. It examines the limitations of exact match, aggregate success, autonomous-submission rates, and self-correction claims. Research on Spider, CoSQL, RAT-SQL, PICARD, cross-domain benchmark analysis, data-quality verification, production-readiness testing, human-AI interaction, data cascades, and model reporting supports a central conclusion: metrics should follow the system boundary and the consequences of error. A mature evaluation program combines offline replay, execution-based tests, human studies, staged production experiments, incident analysis, and post-deployment monitoring, with results stratified by risk and task difficulty.

1. What Does It Mean for an Analytics Agent to Work?

An analytics agent can fail while producing impeccable prose. It can generate SQL that parses but answers the wrong question, select a stale table, omit a policy filter, consume excessive resources, or create a warehouse artifact that breaks when late data arrives. It can also succeed technically while increasing review burden or encouraging users to trust results they do not understand. A single accuracy number cannot represent these outcomes.

The evaluation problem begins with the system boundary. If the evaluated object is a text-to-SQL parser, a query-level measure may be appropriate. If it is a conversational BI service, the boundary includes clarification, schema retrieval, execution, and presentation. If it is a warehouse delivery agent, the boundary includes configuration, code, submission, diagnosis, revision, and human authorization. Metrics that stop before the claimed boundary systematically overstate reliability.

SiriusBI and SiriusDeliver make this progression visible. They can be read not as competitors but as adjacent stages in an analytics value chain. Evaluating them requires a layered account of component capability and whole-workflow consequences.

2. The Benchmark Foundation and Its Limits

Spider transformed text-to-SQL evaluation by placing different databases in training and test splits. Its 10,181 questions and 5,693 unique complex SQL queries across 200 databases made cross-domain schema generalization a first-class challenge (Yu et al., 2018). Exact-set or structural matching on such a benchmark provides a consistent way to compare parsers, and execution accuracy asks whether a candidate query produces the expected result.

Both measures are incomplete. Two SQL programs can be semantically equivalent while differing structurally, so exact match can label a valid alternative as wrong. Execution accuracy can label a wrong

query as correct when a small database happens not to distinguish them. It can also overlook nondeterminism, cost, security, and behavior under future data. Pourreza and Rafiei (2023) show why human evaluation and careful benchmark analysis matter: reference queries are not infallible, and model outputs may sometimes be preferable even when they diverge from the gold form.

Model innovations reveal further dimensions. RAT-SQL improves schema encoding and linking by representing relations among question tokens and schema elements (Wang et al., 2020). PICARD constrains autoregressive decoding through incremental parsing, increasing the likelihood that outputs belong to the valid target language (Scholak et al., 2021). These works motivate diagnostic metrics: schema-link accuracy, valid-SQL rate, repair rate, and error location. A higher final score is useful, but a system builder also needs to know whether failures arise from intent, linking, generation, or execution.

Conversational settings add history. CoSQL contains multi-turn interactions with clarification, unanswerable requests, SQL-grounded dialogue state, and response generation over unseen databases (Yu et al., 2019). Turn-level query accuracy misses whether the system preserves user intent across turns or recovers gracefully from ambiguity. Conversation evaluation should include state consistency, clarification appropriateness, cumulative task success, and the cost imposed on the user.

3. Reading SiriusBI's Evidence

Jiang et al. (2024) present SiriusBI as a comprehensive LLM-powered BI solution for industrial deployment. The architecture coordinates multiple modules, uses semantic completion, knowledge-guided clarification, and proactive querying, and selects between a one-step fine-tuning approach and a two-step semantic-intermediate-representation method according to data conditions. Experiments reportedly cover real-world datasets and public benchmarks, while user studies address productivity and experience. The system is described as serving enterprise clients across finance, advertising, and cloud domains, attaining more than 93 percent SQL-generation accuracy and reducing analyst query time from minutes to seconds.

These claims address several layers: component quality, workflow speed, user response, and deployment breadth. Their value is greater than a benchmark-only result because they show that a coordinated system can operate in real settings. Their interpretation still depends on definitions. "SQL-generation accuracy" might denote exact match, execution equivalence, expert judgment, or a domain-specific acceptance measure. Each answers a different question. Query-time improvement may measure model latency, elapsed task time, or active analyst effort. Deployment across sectors demonstrates diversity but does not reveal the distribution of schema complexity and risk.

A strong replication package would publish the metric definitions, adjudication procedure, sampling frame, exclusion rules, failure taxonomy, and confidence intervals without exposing proprietary data. It would stratify results by familiar versus new domains, query complexity, ambiguity, and generation route. Because SiriusBI dynamically chooses a route, evaluators should report not only route-specific performance but also routing regret: how often the selector chooses a worse method than the best available route for that case.

Clarification requires its own counterfactual. The system should be evaluated on whether it detects consequential ambiguity, asks a discriminative question, incorporates the answer, and avoids unnecessary interruptions. A correct final query after five avoidable questions is not equivalent to the same query after one useful question. Human-AI interaction research emphasizes appropriate expectation-setting, efficient correction, and preserved control (Amershi et al., 2019); those qualities can be operationalized in task studies rather than left as interface aspirations.

4. Reading SiriusDeliver's Evidence

Xie et al. (2026) define a broader operational boundary for SiriusDeliver. The agent retrieves warehouse context, configures workflows, generates code, submits tasks, and diagnoses failures. Its hierarchy orchestrates reusable skills, its lifecycle module verifies and revises artifacts before and after platform execution, and its trace-driven mechanism updates skills from delivery trajectories.

The reported evidence includes offline real-world cases and a large production deployment on Tencent Cloud WeData. Over two months, SiriusDeliver reportedly served 3,600 monthly active users, handled 18,240 delivery sessions, spanned six business teams and four warehouse task types, achieved 87.2 percent end-to-end success, and autonomously submitted 73.5 percent of sessions. A one-month A/B test reduced median delivery time from 228 to 23 minutes and engineer effort from 95 to 11 minutes while maintaining comparable final success (Xie et al., 2026).

These are unusually meaningful systems measures. End-to-end success tests more of the claimed workflow than code compilation does. Engineer effort distinguishes elapsed platform time from scarce human attention. An A/B design is stronger than an uncontrolled before-and-after comparison. Nevertheless, the metrics can interact in ways that complicate interpretation. If the agent handles routine work while humans receive the hardest cases, autonomous-submission and human-effort measures may improve without demonstrating competence on the full workload. If users abandon difficult sessions early, the denominator for end-to-end success matters. Comparable final delivery success can coexist with different downstream defect rates.

Evaluation should therefore report a task funnel. The funnel starts with all requested sessions and records eligibility, abstention, clarification, human takeover, autonomous submission, platform success, post-execution validation, and sustained healthy operation. Each exit needs a reason. Difficulty and risk labels should be assigned independently of the agent's outcome where possible, avoiding a circular definition in which all failures are reclassified as unsupported.

Artifact lifecycle control also creates specific measures: pre-execution defect detection, false alarm rate, defects escaping into execution, successful automatic repair, regression after repair, rollback rate, and time to stable recovery. Automated data-quality verification shows how constraints can be executed at scale (Schelter et al., 2018). Those checks should appear in the evaluation as observable evidence, not be combined invisibly into a single agent score.

5. An Evaluation Matrix

A complete program can organize evidence along six rows and four columns. The rows are intent, artifact, execution, human work, governance, and longitudinal operation. The columns are quality, efficiency, robustness, and accountability. The matrix prevents a team from declaring success because one corner looks strong.

Intent quality includes correct metric definitions, filters, temporal scope, and handling of unanswerable requests. Intent efficiency includes clarification turns and time to a confirmed interpretation. Robustness tests paraphrases, incomplete requests, conflicting definitions, and domain novelty. Accountability requires a record of assumptions and decisions.

Artifact quality includes semantic equivalence, syntax, schema compatibility, tests, and documentation. Efficiency includes generation and review time. Robustness covers schema drift, dialect changes, and adversarial or malformed inputs. Accountability includes provenance, versioning, and ownership. Model-card practice offers a precedent for documenting intended use, evaluation conditions, and limitations rather than treating a score as context-free (Mitchell et al., 2019).

Execution quality includes correct results and successful workflow completion. Efficiency includes latency, compute cost, retries, and resource waste. Robustness includes partial failure, late data, dependency outages,

and retry safety. Accountability includes tool-call logs, policy decisions, and reversible actions. These measures should be computed by the environment whenever possible; an agent's claim that it succeeded is not execution evidence.

Human-work quality includes whether users reach defensible decisions. Efficiency includes active effort, interruption burden, and time to recovery. Robustness covers novice and expert populations, different roles, and accessibility. Accountability includes clear responsibility and usable review information. Satisfaction is informative but insufficient: users can like systems that are confidently wrong.

Governance quality includes access compliance, appropriate approval, data minimization, and policy adherence. Efficiency includes the cost of review and exception handling. Robustness tests prompt injection, privilege boundaries, and policy change. Accountability includes auditability and incident reconstruction. Longitudinal quality includes artifact freshness and sustained correctness; longitudinal efficiency includes maintenance load; robustness includes drift; accountability includes skill and policy version histories.

6. Designing the Evidence Pipeline

Offline evaluation should begin with frozen, versioned cases. Each case includes the request, permitted context, expected constraints, environment snapshot, and adjudication rubric. Query cases should include alternative valid SQL where possible. Delivery cases should include full dependencies and expected postconditions. Leakage checks are essential when language models or retrieved memories may contain benchmark examples.

Replay adds realism. Historical sessions can be rerun against a controlled platform snapshot, preserving tool observations without causing production side effects. Replay is especially useful for failure diagnosis and repair. It must include unsuccessful and escalated cases, not only clean success traces. Data-cascade research shows how upstream data work and failures can be rendered invisible when attention centers on model construction (Sambasivan et al., 2021); representative replay should deliberately recover that neglected work.

Staged online evaluation follows. Shadow mode observes requests and proposes actions without executing them. Canary mode permits bounded actions in a restricted namespace. A randomized trial compares assistance conditions under predefined stopping rules. Production monitoring then tracks outcomes that appear after the immediate task, including data-quality incidents, rollbacks, repeated corrections, and abandoned artifacts.

Readiness gates can borrow from the test-oriented approach of Breck et al. (2017). Instead of one pass threshold, teams specify evidence requirements for each risk tier: test coverage, monitoring, rollback, ownership, security review, and human authorization. The purpose is to expose missing safeguards before scaling a promising model result into a business-critical service.

7. Error Taxonomies and the Problem of Self-Correction

Aggregate metrics conceal what the system learned and what it merely survived. A shared error taxonomy should distinguish request ambiguity, semantic-definition error, schema-link error, invalid syntax, valid-but-wrong query, permission violation, excessive cost, configuration error, dependency failure, data-quality failure, diagnosis error, unsafe repair, and human-interface error. Severity and reversibility should be recorded separately from frequency.

Claims of automatic revision also require external checks. Research on intrinsic self-correction finds that language models do not reliably improve reasoning merely by being asked to review themselves without informative external feedback (Huang et al., 2024). In analytics, correction should be grounded in parser errors, execution traces, data constraints, differential tests, policy results, or human input. An agent generating a second plausible answer is not a verifier.

This point strengthens the SiriusDeliver lifecycle approach: post-execution observations can supply real feedback (Xie et al., 2026). But even external feedback may be ambiguous. A job can succeed while producing semantically wrong rows; a row-count anomaly can be legitimate. Evaluation should identify which feedback signal triggered a revision and whether the repair addressed the underlying fault rather than merely making the error disappear.

8. Reporting Results Without Hiding Tradeoffs

Every headline metric should be accompanied by its denominator, time window, task distribution, uncertainty, and failure exclusions. Median time is valuable for skewed workflows, but tail latency matters for operational burden. Autonomous rate should be paired with escalation quality and risk. End-to-end success should be paired with post-deployment defect escape. User effort should distinguish active work, waiting, and later rework.

Results should also expose tradeoffs. Aggressive clarification may improve semantic accuracy while increasing abandonment. Strict validation may reduce incidents while increasing time to delivery. Broader autonomy may reduce immediate effort while increasing monitoring and maintenance. A Pareto view is more honest than collapsing these outcomes into a weighted score chosen after the experiment.

Finally, reports should document data and context. Datasheet-style practices can disclose evaluation-set origins, composition, collection, exclusions, and known gaps (Gebru et al., 2021). Model-card-style practices can state intended deployment domains, tested capabilities, limitations, and ethical considerations (Mitchell et al., 2019). For proprietary enterprise cases, aggregated distributions and rigorous procedures can improve interpretability without releasing sensitive records.

Conclusion

Analytics-agent evaluation must expand as agent scope expands. Spider, CoSQL, schema-aware parsing, and constrained decoding provide essential component evidence. SiriusBI demonstrates the need to evaluate coordinated, interactive BI behavior and its effect on analyst work. SiriusDeliver demonstrates the need to evaluate complete warehouse delivery, artifact revisions, human effort, and operational outcomes. No one metric travels safely across all these boundaries.

The practical answer is a portfolio: diagnostic benchmarks, execution-based verification, human task studies, controlled production experiments, risk-stratified funnels, and longitudinal incident monitoring. Evidence should come from the system that bears the consequence, and every autonomous claim should identify the unresolved work shifted to people or infrastructure. Moving beyond exact match does not mean abandoning rigor. It means matching rigor to the real object under evaluation.

References

Jiang, J., Xie, H., Yang, J., Shen, S., Wang, Z., Zheng, Y., ... & Jiang, J. (2024). Siriusbi: A comprehensive llm-powered solution for data analytics in business intelligence. *arXiv preprint arXiv:2411.06102*.

Xie, H., Zhou, X., Yang, J., Shen, S., Wang, Z., Zheng, Y., ... & Jiang, J. (2026). SiriusDeliver: Automating Data Warehouse Delivery at Tencent. *arXiv preprint arXiv:2608.09185*.

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., et al. (2019). Guidelines for human-AI interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (pp. 1-13). https://doi.org/10.1145/3290605.3300233

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data* (pp. 1123-1132). https://doi.org/10.1109/BigData.2017.8258038

Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daume III, H., & Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM, 64*(12), 86-92. https://doi.org/10.1145/3458723

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*. https://openreview.net/forum?id=IkmD3fKBPQ

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., et al. (2019). Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (pp. 220-229). https://doi.org/10.1145/3287560.3287596

Pourreza, M., & Rafiei, D. (2023). Evaluating cross-domain text-to-SQL models and benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 1601-1614). https://aclanthology.org/2023.emnlp-main.99/

Sambasivan, N., Kapania, S., Highfill, H., Akrong, D., Paritosh, P., & Aroyo, L. M. (2021). Everyone wants to do the model work, not the data work: Data cascades in high-stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1-15). https://doi.org/10.1145/3411764.3445518

Schelter, S., Lange, D., Schmidt, P., Celikel, M., Biessmann, F., & Grafberger, A. (2018). Automating large-scale data quality verification. *Proceedings of the VLDB Endowment, 11*(12), 1781-1794. https://doi.org/10.14778/3229863.3229867

Scholak, T., Schucher, N., & Bahdanau, D. (2021). PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (pp. 9895-9901). https://doi.org/10.18653/v1/2021.emnlp-main.779

Wang, B., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2020). RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 7567-7578). https://doi.org/10.18653/v1/2020.acl-main.677

Yu, T., Zhang, R., Er, H., Li, S., Xue, E., Pang, B., et al. (2019). CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (pp. 1962-1979). https://doi.org/10.18653/v1/D19-1204

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., et al. (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* (pp. 3911-3921). https://doi.org/10.18653/v1/D18-1425