From Natural-Language Questions to Governed Warehouse Actions: A Systems Review of Enterprise Analytics Agents

Literature review and conceptual analysis

Abstract

Large language models are changing enterprise analytics at two connected but historically separate layers. At the access layer, they translate business questions into queries, clarify intent, and explain results. At the delivery layer, they retrieve engineering context, configure workflows, generate transformation code, submit jobs, diagnose failures, and revise artifacts. This review argues that treating either layer as a stand-alone generation problem understates the central systems challenge: every useful action must remain grounded in schemas, dependencies, policies, and observable execution. SiriusBI and SiriusDeliver illustrate the emerging end-to-end design space. The first organizes conversational business intelligence around coordinated modules, multi-round clarification, and data-conditioned SQL-generation strategies; the second organizes warehouse delivery around hierarchical skill orchestration, lifecycle-aware artifact control, and learning from execution traces. Read alongside work on text-to-SQL, retrieval, tool use, data quality, provenance, and production technical debt, these systems suggest a common architecture built from constrained intent resolution, typed actions, executable verification, and accountable handoffs. The review synthesizes that architecture, identifies where benchmark evidence and production evidence answer different questions, and proposes design principles for analytics agents that can increase speed without obscuring responsibility.

1. Analytics Automation Is a Chain, Not a Prompt

Enterprise analytics is often described as if a single language barrier separates a business user from a useful answer. That description is attractive because modern language models can produce plausible SQL and code from short instructions. In practice, however, an analytic request crosses a chain of representations: an underspecified business question becomes a metric definition, a schema-linked query, an executed result, an interpretation, a transformation task, a scheduled workflow, and finally an operational artifact monitored over time. An error introduced early can survive syntactic checks and acquire organizational authority simply because later stages run successfully.

The research history of text-to-SQL shows why this chain cannot be reduced to fluent generation. Spider deliberately separates training and test databases, forcing systems to generalize to unfamiliar schemas rather than memorize query templates (Yu et al., 2018). RAT-SQL responds by explicitly modeling relations among tables, columns, and question tokens, making schema structure part of the encoder rather than background context (Wang et al., 2020). PICARD adds a different guardrail: it rejects inadmissible tokens during decoding so that a language model cannot freely wander outside the grammar of the target language (Scholak et al., 2021). These advances improve important components, but an enterprise answer can be syntactically valid and still use the wrong definition of revenue, the wrong temporal grain, or a deprecated table.

At the other end of the chain, warehouse delivery is rarely a single code-generation event. A production task may require upstream dependency inspection, partition conventions, compute settings, ownership metadata, approval checks, backfill policy, and post-submission diagnosis. The general lesson resembles the technical-debt analysis of Sculley et al. (2015): the model is only a small part of the production system, while

data dependencies, feedback loops, and glue code dominate long-term risk. Analytics agents therefore need a systems theory that connects language understanding to governed execution.

2. SiriusBI as an Access-Layer System

SiriusBI is useful precisely because it frames business intelligence as more than one-shot SQL synthesis. Jiang et al. (2024) describe an end-to-end, multi-module service intended for industrial deployment. Its three reported design commitments are especially consequential. First, coordination across modules expands the scope from query generation toward a complete BI interaction. Second, a multi-round mechanism combines semantic completion, knowledge-guided clarification, and proactive querying. Third, a data-conditioned strategy selects between an efficient one-step fine-tuning route and a two-step route that uses a semantic intermediate representation for lower-cost domain transfer.

The intermediate-representation choice has a clear intellectual lineage. IRNet separates schema linking, generation of a domain-oriented SemQL representation, and deterministic conversion to SQL (Guo et al., 2019). Such decomposition narrows the gap between business intent and database implementation. SiriusBI extends that intuition from a benchmark parser into an operational selection problem: different data conditions may justify different generation paths. This is important because no single adaptation strategy is uniformly economical. A stable, high-volume domain may reward fine-tuning, whereas a new or sparse domain may benefit from an intermediate representation that externalizes structure.

The system's interaction mechanism also addresses a failure mode that grammar constraints cannot solve. If "active customer" has multiple legitimate definitions, forcing the model to emit valid SQL merely crystallizes ambiguity. Clarification changes the epistemic status of the output: the system exposes uncertainty and acquires missing intent before execution. The official report associates SiriusBI with deployments in finance, advertising, and cloud settings, more than 93 percent SQL-generation accuracy, and a reduction of analyst query time from minutes to seconds (Jiang et al., 2024). These figures provide valuable evidence of practicality. They do not, by themselves, establish that every business definition is correct, that accuracy is comparable across domains, or that downstream decisions are improved. The contribution is best read as a production architecture and deployment case, not as a universal guarantee.

Retrieval is one way to strengthen the knowledge boundary around such a system. Retrieval-augmented generation separates model parameters from an inspectable, updateable memory source (Lewis et al., 2020). In BI, the retrieved objects should be more structured than generic documents: metric contracts, schema descriptions, ownership records, prior approved queries, freshness indicators, and access rules. Retrieval can make context current, but only if the system records what was retrieved, which version was used, and how that evidence influenced the generated query.

3. SiriusDeliver as a Delivery-Layer System

SiriusDeliver begins where many conversational BI demonstrations stop. Xie et al. (2026) characterize warehouse task delivery as a sequence that includes context retrieval, workflow configuration, code generation, platform submission, and failure diagnosis. Their architecture includes a hierarchical delivery agent that orchestrates warehouse skills, an artifact lifecycle control module that verifies and revises artifacts before and after platform execution, and a trace-driven mechanism for maintaining reusable skills from delivery trajectories.

This design shifts the unit of automation from a code snippet to a stateful delivery episode. A hierarchy matters because a warehouse request contains goals at different levels. "Build a daily retention table" is a business objective; choosing sources, defining joins, configuring partitions, registering lineage, submitting the workflow, and responding to errors are operational subgoals. A flat agent may produce locally plausible steps without preserving global invariants. Hierarchical control can assign each step a typed contract and return evidence to a parent controller.

Tool-use research helps explain the promise and the danger of this approach. ReAct interleaves reasoning with actions so that observations from an environment can update a plan (Yao et al., 2023). Toolformer shows that language models can learn when and how to call external tools rather than relying exclusively on parametric knowledge (Schick et al., 2023). A data-platform agent requires both capabilities, but production tools have side effects. A schema lookup is read-only; a task submission may allocate substantial compute, expose data, or overwrite a partition. The action space must therefore carry permissions, dry-run modes, idempotency keys, resource limits, and escalation conditions. General tool competence is not sufficient governance.

The reported SiriusDeliver deployment makes the operational claim concrete. Across a two-month period, the system served six business teams, four warehouse task types, 3,600 monthly active users, and 18,240 delivery sessions. It reports an 87.2 percent end-to-end success rate and a 73.5 percent autonomous submission rate. A one-month A/B test reportedly 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 relevant measures because they capture an entire workflow rather than isolated code correctness. Yet autonomy rate and success rate must be interpreted with the distribution of task difficulty, intervention policy, and excluded cases. A system can appear more autonomous by declining hard tasks or routing them to humans before the measured episode begins.

4. A Shared Architecture Across Access and Delivery

SiriusBI and SiriusDeliver address different stages, but their strongest ideas can be expressed as one control loop. The loop begins with intent acquisition. A user request is decomposed into business semantics, data requirements, and operational constraints. The system then assembles context from governed sources, proposes a typed intermediate artifact, validates that artifact, executes it in a controlled environment, observes results, and either completes, clarifies, repairs, or escalates.

The first architectural principle is to preserve intermediate representations. SQL is one such representation, but metric specifications, dependency graphs, workflow manifests, validation rules, and release records are equally important. They create locations where policies can be checked before effects occur. PICARD demonstrates the value of constraining a decoder with formal admissibility (Scholak et al., 2021); enterprise systems should extend the idea from SQL grammar to organization-specific contracts. A partitioned table definition, for example, can be rejected if it omits retention policy or owner metadata even when the transformation code is valid.

The second principle is execution-grounded verification. A generated query should be parsed, cost-estimated, permission-checked, and, where possible, run on a bounded sample before full execution. A warehouse artifact should pass static checks, dependency resolution, data-contract validation, and a canary run. Automated data-quality verification provides a foundation for expressing constraints as reusable checks rather than informal review comments (Schelter et al., 2018). The important shift is from asking whether text looks correct to gathering evidence about whether an artifact behaves correctly in its intended environment.

The third principle is provenance at both data and action levels. Classical database provenance distinguishes why an output exists from where its values came from (Buneman et al., 2001). An agentic system needs an expanded record: which request initiated the work, which metadata was retrieved, which model and policy versions acted, which tool calls occurred, what observations returned, which human approved the transition, and which artifacts were changed. This action provenance supports incident reconstruction and creates training material, but it must not become an unbounded store of sensitive prompts and data values.

The fourth principle is separation of proposing and authorizing. A language model may propose a query, validation plan, or workflow patch. A deterministic policy layer decides whether the proposed action may proceed. The policy can combine role-based access, data classification, environment, cost, reversibility, and confidence signals. High-impact operations should require explicit approval even when the model's historical

accuracy is high. This separation limits the damage from prompt injection, stale context, and confident semantic mistakes.

5. Evidence Must Follow the System Boundary

Benchmark evaluation is necessary because it enables controlled comparison. Spider measures cross-domain generalization under a defined schema split (Yu et al., 2018), while component studies such as RAT-SQL isolate improvements in schema encoding and linking (Wang et al., 2020). Production evaluation answers a different question: whether an entire sociotechnical workflow becomes faster, safer, or easier under real constraints. The two evidence types should be connected, not substituted for each other.

For the access layer, a credible evaluation portfolio includes exact or structural match, execution correctness, answer equivalence, clarification efficiency, calibration, unsafe-query prevention, latency, and user task success. For the delivery layer, it includes artifact validity, end-to-end completion, autonomous progress, human effort, rollback incidence, failure recovery, time to diagnosis, and post-release data quality. Metrics must be stratified by domain novelty, schema complexity, operation type, and risk class. Aggregate success conceals whether a system performs well only on repetitive, low-risk tasks.

Evaluation should also test the handoff between layers. A BI conversation may reveal a recurring metric that deserves a curated warehouse artifact. Conversely, a newly delivered table changes the context available to future BI queries. Tests should therefore include semantic continuity: does the definition negotiated in conversation survive transformation into the warehouse contract, and does the resulting table remain discoverable with the same meaning? This is where end-to-end architecture earns its name.

6. Organizational Design and Residual Limits

No architecture eliminates the need for institutional ownership. Metric stewards decide canonical meanings; platform teams define safe action interfaces; data owners set access policy; and analysts identify when an answer is useful. The agent can reduce coordination cost by packaging evidence and routing decisions, but it should not silently inherit authority from the fluency of its explanations.

Technical debt remains a central risk. Sculley et al. (2015) warn that system-level dependencies can outgrow the modeled component. For analytics agents, hidden debt may appear as prompt-specific conventions, undocumented tool semantics, retrieval indexes with unclear freshness, brittle adapters, or skills learned from obsolete traces. Every automation shortcut creates a maintenance obligation. The response is not to avoid automation, but to make dependencies explicit and testable.

The target systems also leave open research questions. SiriusBI reports strong deployment outcomes, yet cross-client reproducibility is difficult when schemas, policies, and evaluation data are proprietary (Jiang et al., 2024). SiriusDeliver demonstrates substantial workflow gains, yet its trace-driven evolution raises questions about skill validation, concept drift, privacy, and rollback when a learned procedure deteriorates (Xie et al., 2026). Both systems would benefit from standardized reporting of failure taxonomies, human intervention boundaries, and longitudinal maintenance cost.

7. Conclusion

Enterprise analytics agents should be designed as governed control systems rather than conversational wrappers around code generation. SiriusBI shows how clarification, modular coordination, and data-conditioned generation can improve the path from a question to an executable query. SiriusDeliver shows how hierarchical skills, artifact lifecycle controls, and execution traces can extend automation into warehouse delivery. Together, they define a larger agenda: connect intent to action through explicit representations, current evidence, constrained tools, executable tests, provenance, and accountable authorization.

The most consequential measure of progress will not be how much SQL or workflow code an agent emits. It will be how often the system reaches a correct and useful outcome, how clearly it reveals uncertainty, how safely it handles side effects, and how readily people can understand and reverse what it did. In that sense, the future of LLM-powered analytics depends as much on database and systems engineering as on language modeling.

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*.

Buneman, P., Khanna, S., & Tan, W. C. (2001). Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001* (pp. 316-330). Springer. https://doi.org/10.1007/3-540-44503-X_20

Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J. G., Liu, T., & Zhang, D. (2019). Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 4524-4535). https://doi.org/10.18653/v1/P19-1444

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems, 33*.

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

Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., et al. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems, 36*. https://doi.org/10.52202/075280-2997

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

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., et al. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems, 28*, 2503-2511.

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

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

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