SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

LLM-Assisted Database Migration Intelligence for Enterprise Platforms

Ramakrishna Pittu

Google LLC, USA. ORCID: 0009-0005-3646-8769

Abstract
Enterprise database migration involves substantially more than copying schemas and records between systems. Organizations must translate incompatible data types, SQL dialects, stored procedures, triggers, indexing strategies, security models, and operational workflows while preserving application behavior and data integrity. Conventional migration tools automate common syntactic conversions but rely heavily on deterministic mapping rules, and they typically require substantial manual intervention once source systems contain proprietary features, dynamic SQL, undocumented dependencies, or business logic embedded in procedural code. This paper proposes an LLM-assisted database migration intelligence framework combining large language models, deterministic compatibility rules, retrieval-augmented generation, dependency analysis, and execution-based validation. The framework treats the language model as a reasoning and recommendation layer rather than an autonomous migration authority: it constructs a migration knowledge graph from schemas, database objects, application repositories, workload telemetry, and organizational standards, then classifies migration complexity, recommends target-compatible transformations, produces traceable remediation proposals, and validates generated artifacts through parsing, compilation, testing, data reconciliation, and performance comparison. An illustrative enterprise scenario shows how the architecture can support migration of a legacy transactional platform to a cloud-native relational database, and a proposed evaluation methodology outlines how transformation accuracy, semantic equivalence, migration coverage, engineering effort, and unsafe-generation rates could be measured in future empirical work. Large language models can materially improve migration discovery and engineering productivity, but only within a governed architecture that preserves deterministic validation, human approval, least-privilege access, and complete auditability.

Keywords: Large language models, database migration, enterprise modernization, SQL transformation, schema conversion, retrieval-augmented generation, dependency analysis, human-in-the-loop automation.

1. Introduction

Enterprise databases accumulate technical and organizational complexity over long operating lifetimes. A single production platform can contain thousands of tables, views, indexes, stored procedures, triggers, scheduled jobs, database links, and application-generated queries, many of which depend on proprietary data types, vendor-specific functions, procedural languages, optimizer hints, and security features with no direct equivalent on a target platform. Migrating such a system demands both a technical conversion exercise and an understanding of the business behavior the source platform encodes.

Where source and target features map directly onto one another, conventional migration utilities perform well: converting common numeric types, restating table definitions, and replacing recognizable SQL functions. Effectiveness declines sharply once migration involves dynamic queries, deeply nested procedures, implicit type coercion, application-generated SQL, or target architectures lacking a direct feature equivalent. In practice, it is this residual category, not the bulk of straightforward conversions, that consumes most enterprise migration budgets and schedules.

A complementary set of capabilities comes from large language models, which can interpret unfamiliar procedural code, explain cross-object dependencies, propose candidate transformations, and reason jointly across schemas, documentation, and application repositories in ways rule-based converters cannot. Yet benchmark evaluations of large language models on text-to-SQL tasks report meaningfully lower accuracy once queries move from simplified academic benchmarks toward realistic enterprise schemas, multi-dialect environments, and multi-step operations. On an enterprise benchmark drawn from authentic operational query logs, even a strong contemporary model achieved only 11.4 percent execution accuracy, far below the

same class of model's reported performance on simplified academic text-to-SQL benchmarks (Gao et al., 2023; Chen et al., 2024). The gap matters directly for migration, because a model that performs adequately on an isolated query may still fail once the same construct is embedded inside a stored procedure with dependent triggers and downstream reporting jobs.

There is a further way database migration differs from ordinary text-to-SQL generation: a migration system must preserve existing behavior rather than merely produce an executable statement. An output can be syntactically valid and still incorrect because of differences in null handling, timestamp precision, collation, transaction isolation, sequence behavior, exception handling, or optimizer behavior between source and target engines. Production migration cannot, for this reason, rely on model confidence alone.

What this paper proposes is a hybrid architecture in which large language models assist with discovery, reasoning, classification, and remediation, while deterministic systems retain authority over validation and deployment. Five contributions follow from that architecture. A layered structure combines database metadata, dependency graphs, migration rules, retrieval, and LLM reasoning. A methodology produces evidence-backed transformations with confidence and provenance rather than opaque suggestions. A validation model explicitly separates generation from approval and execution. A proposed evaluation framework measures correctness, productivity, operational risk, and governance for future empirical work. Finally, an illustrative enterprise scenario shows how the framework could support phased modernization of a financial-services transaction-processing platform.

2. Related Work

2.1 Text-to-SQL and enterprise benchmark evaluation

A comprehensive benchmark evaluation of large language models on text-to-SQL generation was conducted by Gao et al. (2023), establishing that model performance is highly sensitive to schema complexity, query structure, and prompting strategy. Chen et al. (2024) extended this line of work with an enterprise-specific benchmark built from authentic operational query logs spanning nineteen business domains across three anonymized data warehouses and found that a leading contemporary model reached only 11.4 percent accuracy on enterprise-realistic tasks despite considerably stronger results on simplified academic benchmarks. This gap is a central motivation for the present framework: a migration system that treats database translation as a generic text-to-SQL problem inherits the same fragility these benchmarks expose and cannot be trusted to operate without deterministic guardrails.

2.2 SQL dialect translation and schema matching

Translating between SQL dialects for cloud migration was examined by Zmigrod, Alamir, and Liu (2024), who documented the syntactic and semantic pitfalls that arise when vendor-specific constructs lack a literal target-side equivalent. A knowledge-graph-based retrieval-augmented generation approach to schema matching, proposed by Ma, Chakrabarti, Khan, and Molnár (2025), showed that grounding a model's schema-mapping decisions in a structured knowledge graph improves precision over ungrounded prompting. Bozdemir and Bilgin (2026) extended this retrieval-grounded line of work with an embeddings-and-vector-store approach to schema retrieval for SQL generation, reporting improved schema-selection accuracy when retrieval is combined with hierarchical clustering rather than fixed selection rules. Taken together, these three studies establish that grounding, whether through knowledge graphs, vector retrieval, or explicit dialect-translation rules, consistently outperforms reliance on a language model's parametric knowledge alone, an architectural premise this paper adopts in Section 3.5. The vector database management systems underpinning this retrieval infrastructure were surveyed by Pan, Wang, and Li (2024), who note that indexing strategy and retrieval latency become first-order design concerns once retrieval moves from a research prototype into a production pipeline.

2.3 Multi-agent orchestration, large-scale code migration, and generation reliability

An evolving multi-agent orchestration framework, in which the coordination topology among cooperating language-model agents adapts during task execution rather than remaining fixed, was proposed by Dang, Qian, and colleagues (2026); the pattern is directly relevant to a migration pipeline where discovery, transformation, and validation responsibilities could plausibly be distributed across specialized agents rather than handled in a single monolithic reasoning step. Belay et al. (2026) reviewed agentic and LLM-based approaches to multimodal anomaly detection, cataloguing the architectural patterns and open reliability challenges relevant to any agentic system, migration pipelines included, that must detect and act on abnormal or unexpected behavior autonomously. At an industrial scale, Ziftci et al. (2025) documented Google's approach to large-scale code migration using large language models across thirty-nine internal migrations and five hundred ninety-

five total submissions; the model generated approximately 74 percent of code changes and 69 percent of edits, cutting estimated migration time by roughly half, yet final acceptance was still routed through human engineering review rather than autonomous deployment, a governance posture consistent with the human-approval layer proposed here in Section 3.8. A complementary problem, LLM-generated documentation during legacy code modernization, was examined by Diggs et al. (2025), who found documentation quality and completeness varied enough across cases that human review remained necessary before the documentation itself could be trusted as migration evidence.

Two further findings motivate this framework's emphasis on deterministic validation. The HalluLens benchmark for large language model hallucination, introduced by Bang et al. (2025), demonstrates that even strong general-purpose models continue to produce fabricated or unsupported content at rates that make unverified model output unsuitable as the sole basis for a production change. How prompt-engineering technique affects the reliability of structured-data extraction from language models was evaluated by Chen et al. (2025), who found extraction accuracy highly sensitive to prompt design, a finding treated here as further justification for the structured, retrieval-grounded prompting approach described in Section 3.5 rather than open-ended prompting. Automated recommendations for evolving relational database schemas, proposed by Etien and Anquetil (2024), offer a deterministic complement to the language-model-driven schema reasoning this paper layers on top of a compatibility rule engine.

2.4 Enterprise database operational practice and governance

The practical implementation of SQL Server high-availability and disaster-recovery configurations using the AlwaysOn feature set without shared storage was documented by Hayat and Soomro (2018), establishing the kind of operational continuity requirement that any migration framework must preserve rather than disrupt. A framework for strengthening data governance in multi-cloud environments was proposed by Akande et al. (2025), addressing security, compliance, and operational resilience concerns that extend directly to migrations moving data across cloud boundaries. Policy-driven automation for scalable governance in enterprise big-data platforms was examined by Thota (2019), an operational governance pattern adapted here for migration-specific approval policies in Section 3.6. Practical strategies for migrating legacy information-management systems to AWS and GCP were documented by Vishnubhatla (2017), illustrating the hybrid-cloud realities an enterprise migration intelligence framework must accommodate rather than assume away. The retrieval-augmented generation approach that grounds this paper's LLM reasoning layer traces back to Lewis et al. (2020), who showed that combining parametric generation with non-parametric retrieval improves factual specificity and provenance for knowledge-intensive tasks, a property directly relevant to migration recommendations that must cite the evidence supporting them rather than assert a transformation on the model's authority alone.

Existing commercial migration products emphasize schema conversion, data movement, and compatibility reporting. What is proposed here is not a replacement for those tools but a contextual reasoning layer placed alongside them, one that analyzes their findings, identifies cross-object implications the tools cannot see individually, recommends remediation for gaps in rule-based coverage, and supports engineers in resolving cases where deterministic mappings are incomplete.

3. Proposed Architecture and Methodology

3.1 Architectural principles

Five principles govern the framework. Evidence before generation requires that every recommendation reference a source artifact, target-platform documentation, or an approved migration rule rather than an unsupported model assertion. Deterministic validation means no output is accepted merely because it appears plausible. Separation of duties keeps generation, validation, approval, and deployment as distinct pipeline stages rather than a single automated action. Human authority reserves final approval of any transformation affecting production behavior for a qualified engineer. Complete traceability requires that every finding and transformation retain its inputs, reasoning context, validation results, and approval history. Table I summarizes how these principles map onto architectural layers and their supporting literature.

3.2 Source discovery and metadata ingestion

Migration evidence is collected, in the first architectural layer, from database catalogs and information schemas; table, view, index, sequence, and constraint definitions; stored procedures, functions, packages, and triggers; scheduled jobs and database links; application SQL and object-relational mappings; representative data-type samples; query workload and execution statistics; security roles and row-level policies; and architecture documents and operational runbooks. Each object is normalized into a common intermediate

representation carrying a stable identifier, type, source location, definition, owner, and dependency set, with sensitive values excluded or masked unless essential for downstream validation.

3.3 Dependency and impact graph

Relationships among database and application objects are constructed by a graph-processing layer, with nodes representing tables, columns, views, routines, reports, services, and scheduled processes and edges representing relationships such as reads, writes, invokes, inherits, references, or depends on. This graph enables impact analysis: when a source data type requires transformation, the framework can identify every stored procedure, application query, report, and integration potentially affected by the change, and it supports migration sequencing by identifying strongly connected components and circular dependencies that an object-by-object conversion approach would miss.

3.4 Compatibility rule engine

Each source object is evaluated by a deterministic rule engine against a versioned compatibility catalog, identifying directly compatible features, features requiring syntactic conversion, features requiring semantic redesign, unsupported target features, potential precision or truncation risks, transaction and locking differences, performance-sensitive constructs, and security-policy mismatches. Every finding carries a severity rating, the affected objects, deterministic evidence, target alternatives, and the validation required before acceptance. Direct mappings convert automatically under this rule engine, while complex findings are routed to the retrieval-grounded reasoning layer described next.

3.5 Retrieval-grounded LLM reasoning

Only the context needed for a specific migration task reaches the language model: relevant source definitions, dependency-graph neighbors, target-version documentation, approved organizational patterns, previously accepted transformations, coding and security standards, and test templates, consistent with the retrieval-grounding approach validated by Lewis et al. (2020), Ma et al. (2025), and Bozdemir and Bilgin (2026), illustrated in Figure 1. A candidate transformation must cite the retrieved sources that justify it. When replacing a proprietary function, for example, the response must identify the source behavior, the target limitation, the selected alternative, the semantic assumptions made, and the tests required to confirm equivalence. Four principal tasks fall within this layer: classification of a finding as syntactic, semantic, architectural, performance-related, or security-related; explanation of the incompatibility and its downstream impact in terms an engineer can act on; transformation, meaning generation of a candidate target implementation or remediation plan; and test generation, meaning proposal of boundary cases and equivalence tests appropriate to the finding.

3.6 Migration planning and confidence scoring

Rather than a single opaque score, each candidate transformation is assigned a multi-dimensional confidence profile incorporating rule coverage, retrieval relevance, dependency completeness, compilation status, test coverage, semantic-equivalence results, human-review status, and performance-validation status. A simple transformation supported by an exact compatibility rule and passing tests may qualify for automated approval within organizational policy, whereas a procedural rewrite touching financial calculations requires domain review regardless of how confident the model appears, consistent with the policy-driven governance automation pattern described by Thota (2019).

3.7 Deterministic validation pipeline

An isolated validation environment receives generated artifacts and covers SQL parsing and static analysis; target-database compilation; dependency resolution; unit and regression testing; source-to-target result comparison; row-count and checksum reconciliation; null, precision, and boundary-value testing; transaction and concurrency testing; query-plan and latency comparison; and security and privilege verification. Validation failures return structured feedback to the remediation layer, as shown by the feedback loop in Figure 1. The language model may generate a revised candidate, but it cannot weaken tests or alter acceptance criteria without explicit human approval.

3.8 Human approval and deployment

A migration workbench presents the original object, the compatibility findings, the proposed transformation, the retrieved evidence, the test results, and the confidence profile together, allowing a reviewer to approve, reject, edit, or request additional analysis from a single view. Approved transformations are committed to version control and promoted through established continuous integration and deployment processes, with production deployment authority remaining outside the language model entirely; rollback scripts and reconciliation plans are generated and reviewed before execution, mirroring the operational-continuity discipline documented for SQL Server AlwaysOn deployments by Hayat and Soomro (2018).

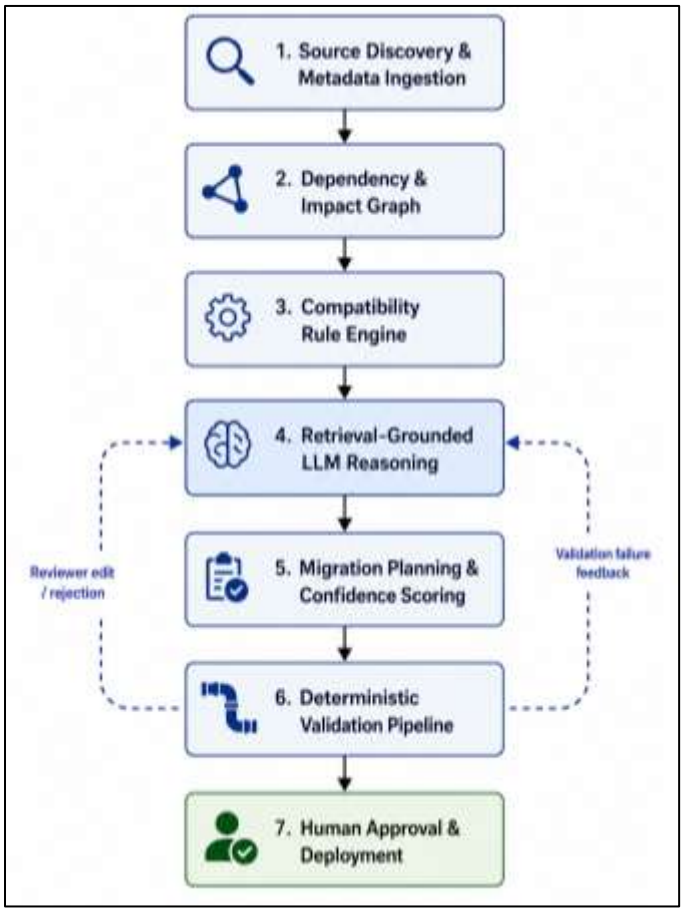

Figure 1. Layered Migration Intelligence Architecture with Validation Feedback Loops.

Architectural Layer	Governance Control	Supporting Literature
Source discovery and ingestion	Data minimization and masking before model exposure	Akande et al. (2025)
Dependency and impact graph	Complete cross-object visibility before any change is approved	Etien and Anquetil (2024)
Compatibility rule engine	Deterministic, versioned evidence for every finding	Zmigrod, Alamir, and Liu (2024)
Retrieval-grounded LLM reasoning	Evidence before generation: citation of retrieved sources	Lewis et al. (2020); Ma et al. (2025); Bozdemir and Bilgin (2026)
Migration planning and confidence scoring	Policy-driven, risk-tiered approval routing	Thota (2019)
Deterministic validation pipeline	No acceptance criteria weakened without human sign-off	Bang et al. (2025)
Human approval and deployment	Human authority over production change and rollback	Hayat and Soomro (2018); Ziftci et al. (2025)

Table I. Architectural Layer, Governance Control, and Supporting Literature.

4. Application to a Representative Enterprise Migration Scenario

The scenario that follows is an illustrative design application of the proposed framework, not an executed empirical study; no production data, client-identifying information, or measured outcomes are reported. Consider a financial-services organization migrating a legacy transaction-processing platform from a proprietary relational database to a cloud-managed, PostgreSQL-compatible platform. The source environment

contains on the order of two thousand tables, thirty-five hundred stored procedures, hundreds of triggers, reporting views, scheduled jobs, and SQL embedded directly in application services.

Direct mappings for standard tables and constraints emerge quickly from an initial rule-based assessment, which also surfaces proprietary procedural packages, autonomous transactions, sequence assumptions, specialized date functions, optimizer hints, database links, and exception-handling behavior with no direct target-side equivalent. From this assessment, the framework builds an inventory and dependency graph and discovers that a settlement procedure invokes multiple packages, updates ledger tables, writes audit events through an autonomous transaction, and is called by three services and an overnight reconciliation job, as shown in Figure 2. Applied object by object, a syntax-only converter would treat each of these components independently and could easily overlook the transaction boundary binding them together.

The relevant procedure definitions, the dependency subgraph, target-platform transaction documentation, and an approved organizational audit pattern all reach the retrieval-grounded reasoning layer, which recommends separating the audit operation into an application-level event written through an outbox table within the primary transaction. This recommendation is classified as an architectural transformation rather than a direct SQL conversion, consistent with the classification task described in Section 3.5. A less complex date-format function elsewhere in the same procedure has a direct target equivalent already identified by the deterministic catalog; the model explains the conversion and generates boundary tests covering time zones, leap years, null values, and invalid inputs, and because compilation and equivalence tests pass, this transformation proceeds through a lower-risk, faster approval workflow, as the two parallel paths in Figure 2 illustrate.

Migration findings are grouped by business capability and dependency order in the workbench, so engineers can prioritize customer onboarding, settlement, reconciliation, and reporting as independently testable domains, with progress measured by validated behavior rather than a raw count of converted objects. During a trial migration, representative workloads are executed against source and target environments by the validation layer, and the resulting data sets are normalized and compared; differences involving decimal rounding and timestamp precision are routed back as semantic defects rather than accepted as passing, and performance regressions are analyzed against target query plans and workload telemetry. What this scenario illustrates is the framework's central value: connecting local, object-level transformations to system-level behavior and giving engineers evidence for deciding whether a given object should be converted, redesigned, retired, or retained temporarily.

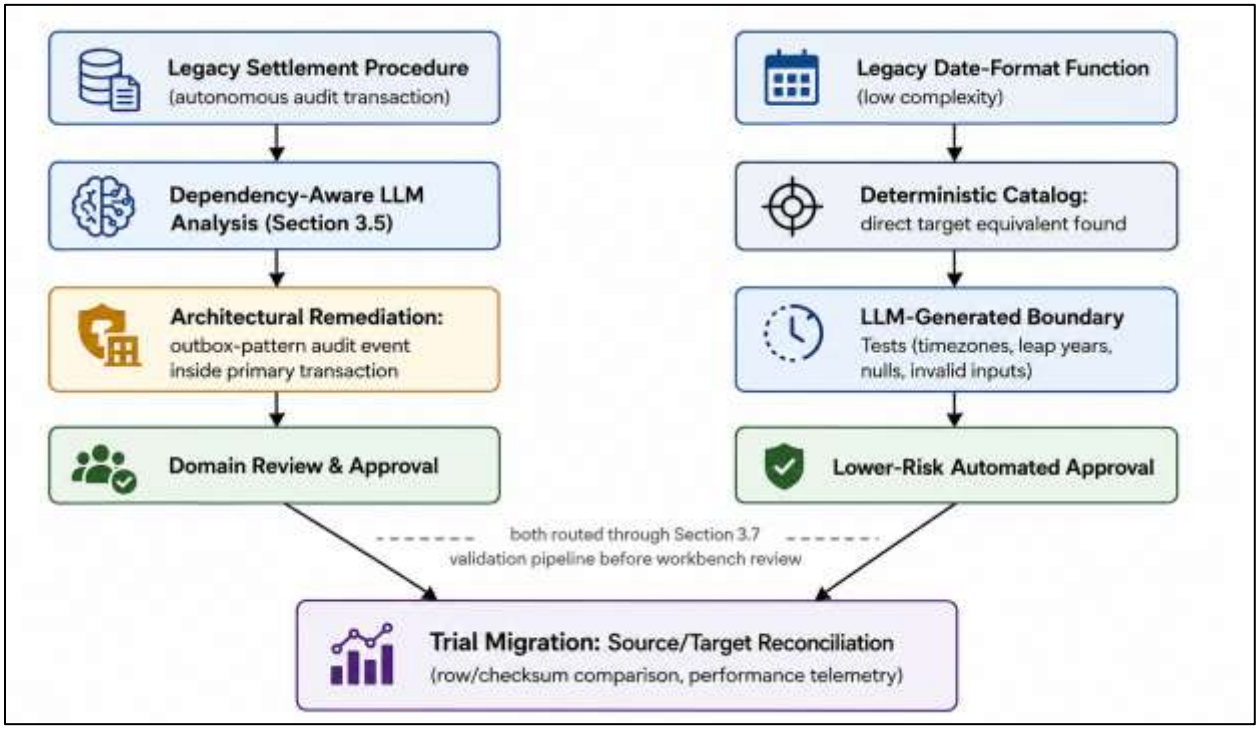

Figure 2. Enterprise Migration Scenario: Architectural vs. Low-Risk Automated Remediation Paths.

5. Evaluation Framework and Discussion

Three conditions should be compared in a rigorous evaluation of the proposed framework: a conventional rule-based migration tool used alone, an LLM-only transformation workflow with no deterministic validation layer, and the proposed hybrid architecture combining rules, retrieval, LLM reasoning, and execution-based validation. Representative enterprise artifacts across a range of difficulty should populate the evaluation dataset, including simple schema definitions, complex queries and views, stored procedures and triggers, dynamic SQL, transaction-sensitive routines, security policies, application-embedded queries, and performance-critical workloads.

Execution success alone is an insufficient signal, since a query can execute and still return incorrect results or exhibit unacceptable performance. Syntax validity, semantic equivalence, operational behavior, and production suitability must therefore be treated as separate outcomes in any credible evaluation, as summarized here in Table II.

Five research hypotheses are proposed for future empirical validation, not reported as established findings. The hybrid framework should improve coverage of complex objects compared with rules alone. Retrieval grounding should reduce unsupported, vendor-specific transformations compared with an LLM-only workflow, consistent with the grounding benefits documented by Ma et al. (2025) and Bozdemir and Bilgin (2026). Decomposed generation and validation should produce higher acceptance rates than single-prompt conversion. Human review time should decrease when recommendations include dependency impact, provenance, and executable tests. And deterministic testing should identify defects that model confidence cannot reliably predict, a concern directly motivated by the hallucination-rate findings of Bang et al. (2025).

Treating autonomous generation as production-ready in a migration context specifically is cautioned against by the accuracy gap Chen et al. (2024) documented between simplified and enterprise-realistic text-to-SQL tasks. The appropriate objective for this framework is not full autonomy but measurable assistance: faster discovery, clearer impact analysis, better first-draft transformations, and more comprehensive tests. Previously completed migration artifacts, used as blinded ground truth with reviewers unaware of whether a given candidate originated from the rule engine, an LLM, or the hybrid system, should anchor a credible future evaluation, with results segmented by object complexity and business criticality so a large volume of simple, successful conversions cannot mask failures concentrated in a smaller number of high-risk routines.

Dimension	Proposed Metric
Conversion coverage	Percentage of objects receiving a usable candidate transformation
Syntax correctness	Percentage of candidates compiling successfully on the target platform
Semantic correctness	Percentage passing source-target equivalence tests
Human acceptance	Percentage approved without substantial rewriting
Engineering efficiency	Review and remediation time per object
Safety	Rate of unsupported or hallucinated transformations
Traceability	Percentage of claims linked to valid, retrievable evidence
Performance	Target latency and throughput relative to source
Operational reliability	Reconciliation defects and rollback events during trial migration

Table II. Proposed Evaluation Framework: Dimensions and Metrics for Future Empirical Validation.

6. Security and Governance

Security must be built into the architecture rather than added afterward, since database migration artifacts often expose sensitive information, including schema structures, customer attributes, access-control definitions, business rules, and infrastructure details. Data minimization, masking, and classification should be applied by the ingestion layer described in Section 3.2 before content reaches the language model, and production records should not be included where synthetic or statistically representative data can support validation instead. Enterprise agreements defining retention, training use, encryption, and geographic-processing requirements should govern model endpoints, consistent with the multi-cloud data-governance concerns raised by Akande et al. (2025).

Least-privilege principles should apply throughout access design: discovery agents should generally operate with read-only metadata permissions, and generated SQL should execute only in isolated non-production environments with resource limits, network restrictions, and approved datasets. Prompt injection is a further

concern the framework must address, since documentation, comments, repositories, and migration tickets are untrusted inputs; retrieved content should be treated strictly as evidence rather than as instructions capable of overriding system policy, with tool permissions enforced outside the model itself. A full audit record should be preserved for every transformation, including model and prompt version, retrieved evidence, source-object version, generated candidate, validation outcomes, human edits, approval identity, and deployment and rollback references. Established public AI risk-management guidance can inform governance of this kind, including the NIST AI Risk Management Framework's lifecycle functions and its Generative AI Profile, which emphasize governance, measurement, testing, transparency, and the management of generative AI-specific risks; these are referenced here as institutional context rather than as sources of empirical claims. Financial calculations, authorization logic, regulatory retention, encryption, and transaction-control routines, being high-risk objects, should never be approved through model confidence alone and should retain accountable human ownership at every stage, consistent with the policy-driven governance-automation approach described by Thota (2019).

7. Limitations and Future Work

Several limitations should temper how the framework's claims are read. Semantic equivalence is difficult to prove conclusively when source behavior is undocumented or depends on production-only data that cannot ethically or practically be reproduced in a validation environment. Dependency graphs can be left incomplete by dynamic SQL and runtime-generated object names, since the graph-construction approach in Section 3.3 depends on discoverable, largely static relationships. Because target performance cannot be predicted reliably from syntax alone, performance regressions may only surface during trial migration rather than earlier design review. Documentation currency and correct versioning are prerequisites for retrieval quality, a dependency the schema-retrieval work of Bozdemir and Bilgin (2026) and Ma et al. (2025) also identifies as a practical constraint rather than a solved problem. And human review can become a bottleneck if confidence routing, described in Section 3.6, is poorly calibrated, sending too many low-risk transformations to manual review or, conversely, too many consequential ones to automated approval.

Migration-specific benchmarks containing paired source and target implementations, test suites, workload traces, and human remediation histories deserve future investigation, comparable in spirit to the enterprise benchmarking approach of Chen et al. (2024) but purpose-built for migration rather than query generation. Graph-enhanced retrieval, automated invariant discovery, transaction-semantics comparison, and performance-aware generation all warrant additional work. Responsibilities among discovery, transformation, testing, security, and performance agents may eventually be separated by multi-agent architectures, as explored by Dang et al. (2026) and Belay et al. (2026) for anomaly detection; however, adding more agents does not eliminate the need for the deterministic controls described in Sections 3.4 and 3.7. Future systems should also support continuous migration intelligence that can detect compatibility regressions as source applications evolve over long modernization programs, extending the automated schema-evolution recommendations proposed by Etien and Anquetil (2024) from a single migration event to an ongoing monitoring posture.

8. Conclusion

Large language models can provide meaningful intelligence for enterprise database migration, particularly in discovery, dependency interpretation, incompatibility explanation, candidate transformation, and test generation. On the evidence reviewed in this paper, however, they should not be trusted to independently determine whether a migration is correct or safe. A production-capable architecture must combine LLM reasoning with deterministic rules, retrieval grounding, dependency graphs, isolated execution, semantic validation, performance testing, human approval, and complete auditability. This hybrid approach positions the language model as an engineering accelerator while preserving the controls enterprise platforms require. The most valuable outcome of such a system is not the maximum volume of automatically generated code. It is a migration process in which every transformation is explainable, testable, traceable, and operationally defensible.

Ethics Statement

This work does not involve human participants, personally identifiable data, or animal subjects. The enterprise migration scenario in Section 4 is illustrative and design-oriented; no production data, client-identifying information, or proprietary organizational material was used or disclosed.

CRediT Author Statement
Ramakrishna Pittu: Conceptualization, methodology, investigation, writing original draft, reviewing and editing.

Acknowledgments
No external funding was received for this work.

References

1. Akande, N. A., Eziokwu, U. J., Cynthia, U., Ogunsanya, V. A., and Oyebode, D. F. (2025). Strengthening data governance in multi-cloud environments: A framework for security, compliance, and operational resilience. International Journal of Multidisciplinary and Current Educational Research, 7(6), 71-83.
2. Bang, Y., Ji, Z., Schelten, A., Hartshorn, A., Fowler, T., Zhang, C., Cancedda, N., and Fung, P. (2025). HalluLens: LLM hallucination benchmark. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 24128-24156).
3. Belay, M. A., Haghipour, A., Rasheed, A., and Rossi, P. S. (2026). Agentic and LLM-based multimodal anomaly detection: Architectures, challenges, and prospects. Sensors, 26(8), 2330.
4. Bozdemir, M., and Bilgin, M. (2026). Schema retrieval with embeddings and vector stores using retrieval-augmented generation and LLM-based SQL query generation. Applied Sciences, 16(2), 586.
5. Chen, L.-C., Weng, H.-T., Pardeshi, M. S., Chen, C.-M., Sheu, R.-K., and Pai, K.-C. (2025). Evaluation of prompt engineering on the performance of a large language model in document information extraction. Electronics, 14(11), 2145.
6. Chen, P. B., Yang, D., Li, W., Wenz, F., Zhang, Y., Tatbul, N., Cafarella, M., Demiralp, C., and Stonebraker, M. (2024). BEAVER: An enterprise benchmark for text-to-SQL. arXiv preprint arXiv:2409.02038.
7. Dang, Y., Qian, C., Luo, X., Fan, J., Xie, Z., Shi, R., Chen, W., et al. (2026). Multi-agent collaboration via evolving orchestration. Advances in Neural Information Processing Systems, 38, 165025-165059.
8. Diggs, C., Doyle, M., Madan, A., Scott, E. O., Escamilla, E., Zimmer, J., Nekoo, N., et al. (2025). Leveraging LLMs for legacy code modernization: Evaluation of LLM-generated documentation. In 2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code) (pp. 177-184). IEEE.
9. Etien, A., and Anquetil, N. (2024). Automatic recommendations for evolving relational databases schema. arXiv preprint arXiv:2404.08525.
10. Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., and Zhou, J. (2023). Text-to-SQL empowered by large language models: A benchmark evaluation. arXiv preprint arXiv:2308.15363.
11. Hayat, Z., and Soomro, T. R. (2018). Implementation of Microsoft SQL Server using 'AlwaysOn' for high availability and disaster recovery without shared storage. International Journal of Experiential Learning and Case Studies, 3(1), 9-17.
12. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.
13. Ma, C., Chakrabarti, S., Khan, A., and Molnar, B. (2025). Knowledge graph-based retrieval-augmented generation for schema matching. arXiv preprint arXiv:2501.08686.
14. Pan, J. J., Wang, J., and Li, G. (2024). Survey of vector database management systems. The VLDB Journal, 33(5), 1591-1615.
15. Thota, M. R. (2019). Policy-driven automation for scalable governance in enterprise big data platforms. International Journal of Scientific Research and Engineering Trends, 5(6).
16. Vishnubhatla, S. (2017). Migrating legacy information management systems to AWS and GCP: Challenges, hybrid strategies, and a dual-cloud readiness playbook. SSRN.
17. Ziftci, C., Nikolov, S., Sjovall, A., Kim, B., Codecasa, D., and Kim, M. (2025). Migrating code at scale with LLMs at Google. In Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (pp. 162-173).
18. Zmigrod, R., Alamir, S., and Liu, X. (2024). Translating between SQL dialects for cloud migration. In Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (pp. 189-191).