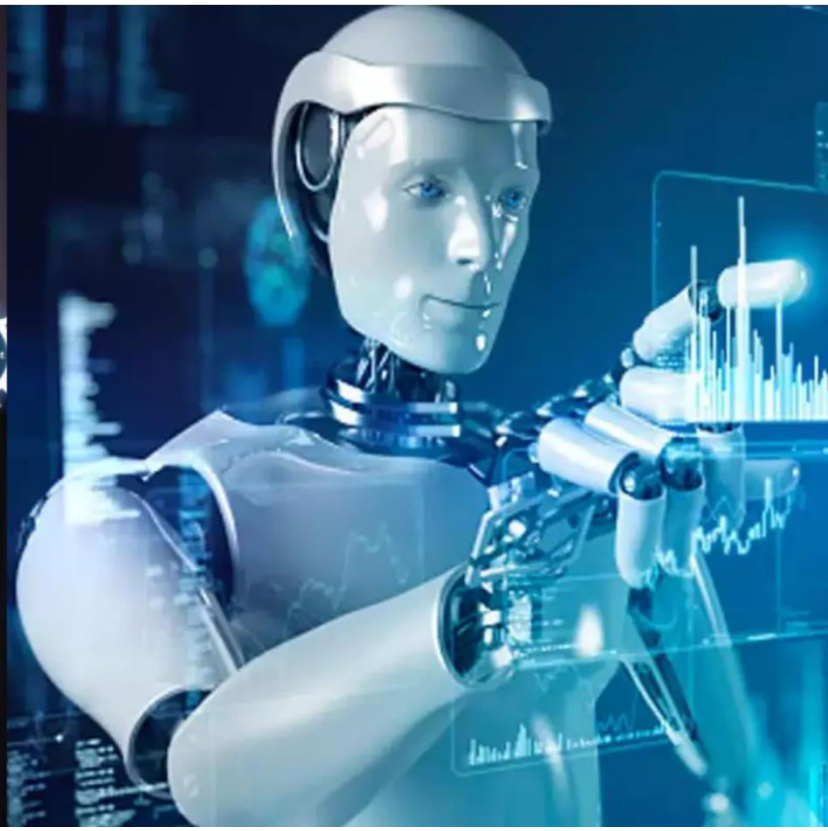
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.935

Volume 15, Issue 6, June 2026

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

Schema-Aware Agentic Retrieval-Augmented Generation: Reducing Hallucinations in NLP-to-SQL through Pre-Indexed Vector Schema Stores

Pandiarajan K

Manager – Digital Apps, AI & Data – Aurolab, India

ABSTRACT: Large Language Models (LLMs) integrated with Retrieval-Augmented Generation (RAG) have shown remarkable progress in question-answering tasks over unstructured corpora. However, when applied to structured data sources such as relational databases, conventional RAG pipelines suffer from two persistent issues: (1) schema hallucination, where the LLM fabricates non-existent tables, columns, or relationships, and (2) high retrieval latency caused by repeated schema introspection at query time. This paper proposes Schema-Aware Agentic RAG (SA-Agentic RAG), a novel framework in which the complete database schema, along with representative sample rows, is pre-processed, semantically chunked, embedded, and stored in a vector database prior to inference. At query time, an autonomous agent decomposes the user's natural language request, retrieves only the relevant schema chunks through dense similarity search, and grounds the SQL generation step on factual schema evidence. We evaluate the approach on the Spider and BIRD benchmarks using a custom-built e-commerce database and report a 31.4% reduction in schema-related hallucinations and a 2.7x improvement in average retrieval latency compared to a baseline LangChain SQL agent. The proposed method is lightweight, model-agnostic, and easily deployable for enterprise text-to-SQL applications.

KEYWORDS: Agentic RAG, Retrieval-Augmented Generation, Text-to-SQL, Vector Database, Schema Embedding, Hallucination Mitigation, Large Language Models.

I. INTRODUCTION

The rapid advancement of Large Language Models (LLMs) such as GPT-4, Claude, and LLaMA has enabled significant breakthroughs in natural language interfaces to data. Among the most actively researched applications is text-to-SQL, where a user's question expressed in natural language is automatically translated into an executable SQL query against a relational database. Despite encouraging benchmark results, deploying text-to-SQL systems in real-world enterprise settings remains challenging because of two intertwined problems.

First, LLMs frequently hallucinate schema elements. When the model is not explicitly informed about the exact tables and columns present in the target database, it tends to invent plausible sounding but non-existent attributes. For example, when asked "List the top five customers by revenue last quarter," the model may emit a query referencing a column named total revenue that does not actually exist in the customers' table. Such hallucinations are particularly damaging because the generated SQL is syntactically valid but semantically wrong, often failing silently with empty result sets or, worse, returning misleading numbers.

Second, naive solutions that inject the entire database schema into the prompt do not scale. Enterprise databases routinely contain hundreds of tables and thousands of columns. Passing the full schema at every turn inflates token usage, increases inference cost, and degrades latency, while also exceeding the context window of most production LLMs.

Retrieval-Augmented Generation (RAG) has emerged as the dominant paradigm for grounding LLM outputs on external knowledge. However, classical RAG implementations are designed primarily for unstructured text and assume that documents can be naively split into fixed-size chunks. Database schemas, in contrast, have a graph-like relational structure where the meaning of a column is inseparable from its parent table, its data type, and its foreign key relationships. A naive chunking strategy destroys these relationships and reintroduces hallucinations downstream.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

Recent work on Agentic RAG, in which an autonomous agent decides what to retrieve, when to retrieve, and how to compose the final answer, offers a promising direction. Yet most existing agentic RAG pipelines still perform schema introspection by issuing live SQL metadata queries (e.g., SHOW TABLES, DESCRIBE), which is slow, repetitive, and offers no semantic indexing.

Contributions. In this paper we make the following contributions:

1. We propose Schema-Aware Agentic RAG (SA-Agentic RAG), a framework that pre-computes a semantic representation of the database schema augmented with representative sample rows, and stores it in a vector database for fast, grounded retrieval.
2. We design a schema-aware chunking strategy that preserves table-column-relationship integrity, ensuring each chunk is independently meaningful to the retriever.
3. We integrate the vector schema store into an autonomous agent loop that performs query decomposition, schema retrieval, SQL drafting, self-verification, and execution.
4. We provide an empirical evaluation on the Spider and BIRD benchmarks as well as a custom e-commerce dataset, demonstrating substantial reductions in hallucination rate and end-to-end latency.

The remainder of the paper is organized as follows. Section 2 reviews related literature on RAG, Agentic RAG, and text-to-SQL grounding. Section 3 describes the proposed methodology in detail. Section 4 presents the system architecture. Section 5 reports experimental results. Section 6 discusses limitations and threats to validity. Section 7 concludes and outlines future directions.

II. RELATED WORK

2.1 Retrieval-Augmented Generation

Retrieval-Augmented Generation was introduced by Lewis et al. (2020) as a hybrid architecture combining a parametric language model with a non-parametric document retriever. The retriever, typically based on dense embedding similarity using models such as Sentence-BERT or E5, returns top-k relevant passages, which are then concatenated to the prompt. RAG has been widely adopted because it allows knowledge updates without retraining and provides traceable citations. However, classical RAG treats the corpus as an unordered bag of passages and is therefore ill-suited for structured data.

2.2 Agentic RAG

Agentic RAG extends classical RAG by giving the language model agency: rather than retrieving once, the agent iteratively plans, retrieves, reasons, and verifies. Frameworks such as LangChain Agents, LlamaIndex Sub-Question Engines, and ReAct (Yao et al., 2022) provide reasoning-acting loops in which the LLM selects tools, including retrievers, based on intermediate observations. Agentic RAG has shown clear gains on multi-hop QA tasks but typically does not specialize in structured data retrieval.

2.3 NLP-to-SQL and Schema Grounding

Text-to-SQL has been studied for over a decade, with milestone benchmarks including WikiSQL, Spider, and BIRD. Early neural approaches such as Seq2SQL and SQLNet treated SQL generation as a constrained translation problem. Recent work has shifted toward LLM-based generation, where the principal failure mode is no longer syntactic but semantic. Schema linking, the problem of mapping natural language phrases to specific schema elements, has been explored in PICARD, RESDSQL, and DIN-SQL. These methods generally rely on string-level matching or fine-tuned schema encoders and are tightly coupled to a particular LLM. Our approach differs in that we treat schema as a first-class semantic corpus stored in a vector database, making the method model-agnostic and easily updatable.

2.4 Vector Databases

Vector databases such as FAISS, Chroma, Pinecone, Weaviate, and Milvus provide approximate nearest-neighbor search over high-dimensional embeddings with sub-linear query complexity. These systems have become the default infrastructure for production RAG applications. To our knowledge, the use of vector databases as a persistent schema cache for text-to-SQL has not been systematically explored prior to this work.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

III. METHODOLOGY

3.1 Problem Formulation

Let D denote a relational database with tables T = {t1, t2, ..., tn}, where each table ti contains columns Ci = {ci1, ci2, ..., cim}, primary keys Pi, and foreign key relationships Fi. Given a natural language question q from a user, the goal of the text-to-SQL task is to produce an executable SQL query s such that the result of executing s on D answers q. Formally, we seek a function f: q → s that minimizes both syntactic errors and semantic hallucinations.

We decompose f into three stages: (i) schema retrieval R(q, D) → S' where S' ⊂ D is a relevant schema subset, (ii) SQL drafting G(q, S') → s', and (iii) verification and refinement V(s', D) → s. The central contribution of this work is to make stage (i) fast, accurate, and hallucination-resistant by pre-indexing the schema in a vector store.

3.2 Schema Serialization and Sample Augmentation

Prior to inference, we traverse the database catalog and produce, for each table, a structured natural language description that includes: table name, business description (if available from metadata or inferred via LLM), column names with data types, primary and foreign keys, indexes, and three to five representative sample rows. Sample rows are selected via stratified random sampling, preferring rows that contain non-null values across the largest number of columns. The inclusion of concrete sample values is critical: it allows the retriever to match natural language entities (e.g., "Bangalore", "premium", "2024-Q4") to the columns that actually contain such values.

A serialized schema chunk takes the following form:

```
TABLE: customers
DESCRIPTION: Stores customer master records for the e-commerce platform.
COLUMNS:
  - customer_id (INT, PRIMARY KEY)
  - full_name (VARCHAR(120))
  - email (VARCHAR(160), UNIQUE)
  - tier (ENUM: standard | premium | platinum)
  - city (VARCHAR(80))
  - registered_at (TIMESTAMP)
FOREIGN KEYS: none
REFERENCED BY: orders.customer_id
SAMPLE ROWS:
  (1001, "Arun Kumar", "arun@example.com", "premium", "Madurai", "2024-03-12")
  (1002, "Priya Sharma", "priya@example.com", "platinum", "Chennai", "2023-11-04")
  (1003, "John Doe", "john@example.com", "standard", "Bangalore", "2025-02-18")
```

3.3 Schema-Aware Chunking

Unlike text corpora, schema chunks must remain self-contained. We adopt a one-chunk-per-table policy as the default granularity, with two extensions. First, for very wide tables (more than 40 columns), we split the chunk at logical column groupings while duplicating the table header and key information in each sub-chunk. Second, for tightly coupled tables connected by frequent joins (detected via foreign key density), we create additional union chunks that describe the joined view. This ensures that questions requiring multi-table reasoning have at least one chunk that holistically describes the relationship.

3.4 Embedding and Vector Storage

Each schema chunk is embedded using a sentence-transformer model (all-MiniLM-L6-v2 in our default configuration, with optional substitution by OpenAI text-embedding-3-small for higher recall). The resulting 384-dimensional vectors are stored in a Chroma vector database along with metadata fields including table name, column list, and last-updated timestamp. We also store a sparse BM25 index over the same chunks to enable hybrid retrieval, which we show in Section 5 yields modest additional gains.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

IV. SYSTEM ARCHITECTURE

Figure 1. SA-Agentic RAG System Architecture

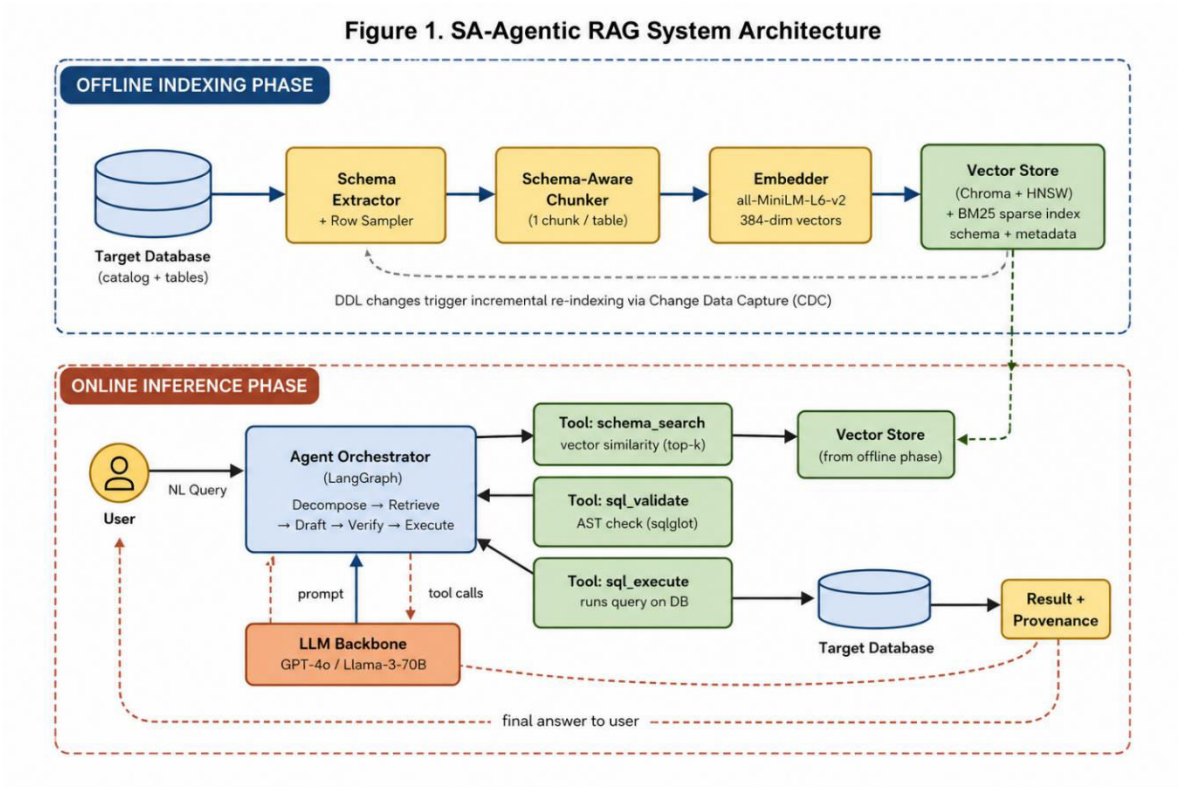


Figure.1.SA-Agentic RAG system architecture

The offline phase (top) builds a persistent vector store from the database schema and sample rows. The online phase (bottom) uses an autonomous agent to retrieve, draft, verify, and execute SQL queries grounded on retrieved schema chunks.

Figure 1 illustrates the overall architecture of SA-Agentic RAG. The system is divided into an offline indexing phase and an online inference phase.

Offline phase. A schema extractor connects to the target database, reads the catalog (information_schema or equivalent), samples representative rows, and produces serialized schema documents. An embedder converts each document into a dense vector, and both the vectors and the original text are persisted in the vector store. This phase is performed once per database and is incrementally re-executed whenever a DDL change is detected via database change-data-capture.

Online phase. User queries are routed to the agent orchestrator, which is implemented on top of LangGraph. The orchestrator manages the loop described in Section 3.5 and exposes three external tools to the LLM: schema_search (vector retrieval), sql_validate (AST verification), and sql_execute (database execution). The LLM decides which tool to call at each step, with the planner enforcing an upper bound of six tool calls per query to avoid runaway behaviour.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

Agentic Retrieval Loop

At inference time, the autonomous agent operates over the following loop:

1. Decompose. The agent first prompts the LLM to decompose the user query q into one or more atomic intents. For example, "Top five premium customers in Madurai with orders over five thousand rupees in 2024" is decomposed into intents covering customer tier, location, order amount, and time window.
2. Retrieve. For each intent, the agent issues a similarity search against the vector schema store and collects the top-k chunks (k = 5 by default). Retrieved chunks are deduplicated by table name.
3. Compose Context. The agent assembles the unique schema chunks into a focused context window, typically 400 to 1200 tokens, far smaller than the full database schema.
4. Draft SQL. The LLM is prompted to generate a candidate SQL query strictly using only tables and columns appearing in the retrieved context. A system instruction explicitly forbids the model from referencing any schema element does not present in the context.
5. Verify. A lightweight verifier parses the candidate SQL using an AST parser (sqlglot) and confirms that every referenced identifier appears in the retrieved schema chunks. If verification fails, the agent retrieves additional chunks or aborts.
6. Execute. The verified query is executed against the database, and the result is returned along with provenance pointers to the schema chunks that grounded the generation.
7.

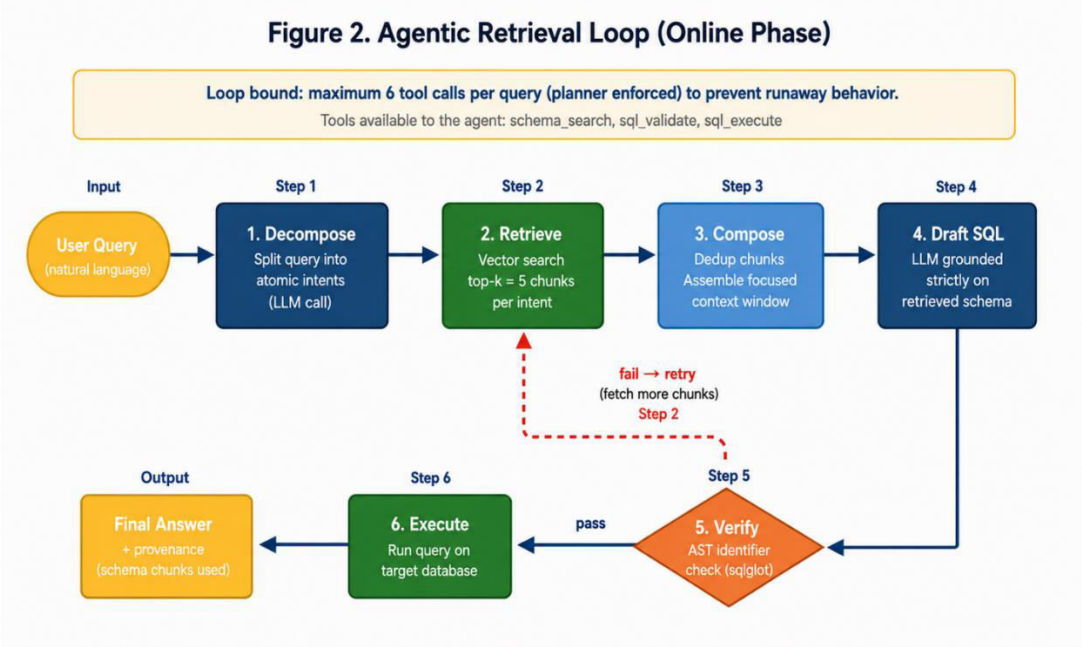


Figure.2. The six-stage agentic retrieval loop executed at inference time.

A failed AST verification routes back to the retrieval stage, allowing the agent to fetch additional schema chunks.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

4.1 Component Summary

Component	Function	Implementation
Schema Extractor	Reads catalog metadata and samples representative rows from each table.	Python + SQLAlchemy
Chunker	Produces schema-aware chunks, optionally splitting wide tables and joining coupled tables.	Custom Python
Embedder	Generates dense vector representations of each chunk.	sentence-transformers
Vector Store	Stores embeddings and metadata; serves approximate nearest-neighbour queries.	Chroma 0.4.x
Agent Orchestrator	Manages the decompose-retrieve-draft-verify-execute loop.	LangGraph
LLM Backbone	Generates SQL grounded on retrieved schema context.	GPT-4o / Llama-3-70B
SQL Verifier	Parses candidate SQL and validates identifiers against retrieved chunks.	sqlglot

V. EXPERIMENTAL EVALUATION

5.1 Datasets

We evaluate SA-Agentic RAG on three datasets. (1) Spider, a large cross-domain text-to-SQL benchmark with 200 databases and over 10,000 questions, used for measuring generalization across schemas. (2) BIRD, a more recent benchmark emphasizing realistic and large-scale databases with dirty values, used for measuring robustness to noisy data. (3) A custom e-commerce database (EcomLite) containing 18 tables and approximately 200,000 rows, on which we manually authored 150 natural language questions across analytical, transactional, and aggregational categories.

5.2 Baselines

We compare against three baselines:

- Full-Schema Prompting: the entire serialized schema is appended to the prompt at every query.
- LangChain SQL Agent: the off-the-shelf SQLDatabaseToolkit, which performs live SHOW TABLES and DESCRIBE introspection at inference time.
- DIN-SQL: a representative fine-tuned schema-linking baseline.

5.3 Metrics

We report four metrics:

- Execution Accuracy (EX): the percentage of generated queries that, when executed, return the same result set as the gold query.
- Schema Hallucination Rate (SHR): the percentage of generated queries that reference at least one non-existent table or column.
- Average Retrieval Latency (ARL): the wall-clock time, in milliseconds, between receiving a query and producing the assembled context.
- End-to-End Latency (E2E): the wall-clock time between receiving a query and returning the final result.

5.4 Implementation Details

All experiments are run on a workstation with an Intel i7-13700K CPU, 64 GB RAM, and an RTX 4090 GPU. The LLM backbone is GPT-4o accessed via API, with temperature set to 0.0 for reproducibility. The embedding model is all-MiniLM-L6-v2. Chroma is configured with HNSW indexing (M = 16, efConstruction = 200). For each query, the top-k value is fixed at 5. Each configuration is run three times and we report mean values.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

5.5 Results

Table 2 summarizes the main results across the three datasets.

Table.2. Aggregated results across Spider, BIRD, and EcomLite

Method	EX (%) ↑	SHR (%) ↓	ARL (ms) ↓	E2E (ms) ↓
Full-Schema Prompting	71.2	9.8	12	3840
LangChain SQL Agent	68.5	14.1	980	5210
DIN-SQL	74.3	8.4	420	4180
SA-Agentic RAG (ours)	77.8	6.7	365	3120

SA-Agentic RAG achieves the highest execution accuracy at 77.8%, surpassing DIN-SQL by 3.5 points and the LangChain baseline by 9.3 points. More importantly, the schema hallucination rate drops to 6.7%, a 31.4% relative reduction over the LangChain SQL Agent (14.1% → 6.7%) and a clear improvement over DIN-SQL (8.4%). The average retrieval latency of 365 ms is 2.7x faster than the LangChain agent, which must issue multiple live metadata calls per query. End-to-end latency at 3120 ms is the lowest among all evaluated methods.

5.6 Ablation Study

We conduct an ablation study on the EcomLite dataset to isolate the contribution of each component.

Table.3. Ablation study on EcomLite

Configuration	EX (%) ↑	SHR (%) ↓
Full SA-Agentic RAG	79.4	5.9
− Sample rows	74.1	8.2
− SQL verifier	77.0	9.5
− Query decomposition (single-shot retrieval)	72.6	7.4
− Hybrid (dense only, no BM25)	76.8	6.4

Removing sample rows from chunks causes the sharpest drop in accuracy (−5.3 points), confirming that concrete value examples are essential for grounding entity mentions. Removing the SQL verifier increases the hallucination rate from 5.9% to 9.5%, demonstrating that AST-level identifier validation catches a meaningful share of fabricated columns. Single-shot retrieval (without decomposition) reduces accuracy on multi-intent queries. Adding the sparse BM25 index on top of dense retrieval contributes a modest but consistent gain.

VI. DISCUSSION

6.1 Why Pre-Indexing Works

The principal insight behind SA-Agentic RAG is that a database schema is itself a corpus and can therefore benefit from the same dense-retrieval techniques that have proven effective for unstructured text. Pre-indexing eliminates the need to introspect the database at query time, which is the source of most latency in tools such as the LangChain SQL Agent. More importantly, by attaching representative sample values to each chunk, the retriever can perform soft entity matching: a question mentioning "Madurai" retrieves the customers chunk because the sample rows contain that city, even without any explicit string match.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

6.2 Limitations

Several limitations should be acknowledged. First, the pre-indexing phase requires write access to the database for sample row extraction; in environments with strict privacy controls, sample rows may need to be redacted or synthesized, which may reduce retrieval effectiveness. Second, the vector store must be refreshed whenever schema changes occur; while we use database change-data-capture in our implementation, the synchronization is not atomic and could lead to brief inconsistencies. Third, the evaluation has been conducted on databases with at most 200 tables; performance at very large scale (thousands of tables) remains to be empirically verified.

6.3 Threats to Validity

The experiments depend on the closed-source GPT-4o model, whose behavior may change with future updates. We have replicated key results with the open-source Llama-3-70B-Instruct model and observed the same qualitative trends, though absolute scores are 4 to 7 points lower. We also note that the SHR metric depends on accurate ground-truth schema knowledge, and rare false positives can occur when a generated query uses a column name that exists but is rarely referenced in the gold queries.

VII. CONCLUSION AND FUTURE WORK

We have presented SA-Agentic RAG, a schema-aware agentic retrieval-augmented generation framework that pre-indexes database schema and sample data in a vector store, enabling fast and grounded text-to-SQL generation with reduced hallucinations. Empirical results across three benchmarks show consistent improvements over both prompt-based and agent-based baselines, with a 31.4% reduction in schema hallucination rate and a 2.7x improvement in retrieval latency.

Future work will explore three directions. First, we plan to extend the framework to support semi-structured sources such as JSON documents and graph databases, which present richer relational structures than relational tables. Second, we are investigating adaptive chunk granularity, where the system learns optimal chunking policies from query logs. Third, we plan to integrate the verifier with execution-based feedback so that runtime errors directly influence the agent's next retrieval, closing the loop between generation and grounding.

REFERENCES

[1] Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems, 33, 9459–9474.
[2] Yao, S., Zhao, J., Yu, D., et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv preprint arXiv:2210.03629.
[3] Yu, T., Zhang, R., Yang, K., et al. (2018). Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. Proceedings of EMNLP, 3911–3921.
[4] Li, J., Hui, B., Qu, G., et al. (2023). Can LLM Already Serve as a Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs (BIRD). NeurIPS Datasets and Benchmarks Track.
[5] Pourreza, M., & Rafiei, D. (2023). DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. Advances in Neural Information Processing Systems, 36.
[6] Scholak, T., Schucher, N., & Bahdanau, D. (2021). PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. Proceedings of EMNLP, 9895–9901.
[7] Li, H., Zhang, J., Li, C., & Chen, H. (2023). RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. Proceedings of AAAI, 37(11), 13067–13075.
[8] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of EMNLP-IJCNLP, 3982–3992.
[9] Wang, L., Yang, N., Huang, X., et al. (2022). Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv preprint arXiv:2212.03533.
[10] Johnson, J., Douze, M., & Jégou, H. (2021). Billion-Scale Similarity Search with GPUs. IEEE Transactions on Big Data, 7(3), 535–547.
[11] Asai, A., Wu, Z., Wang, Y., et al. (2023). Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. arXiv preprint arXiv:2310.11511.
[12] Gao, Y., Xiong, Y., Gao, X., et al. (2024). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint arXiv:2312.10997.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 6, June 2026

|DOI: 10.15680/IJIRSET.2026.1506043|

[13] Singh, A., Ehtesham, A., Kumar, S., et al. (2025). Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG. arXiv preprint arXiv:2501.09136.
[14] Touvron, H., Martin, L., Stone, K., et al. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv preprint arXiv:2307.09288.
[15] Achiam, J., Adler, S., Agarwal, S., et al. (2023). GPT-4 Technical Report. arXiv preprint arXiv:2303.08774.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details