

Business Analytics through Schema-Aware Query Validation

Vedant Patil¹, Mrs. Juhi Agrawal², Sahil Lahudkar³, Sarthak Mahinge⁴

^{1,2,3,4}Department of Information Technology, G H Raisoni College of Engineering and Management, Pune
vedant.patil.it@ghrcemp.raisoni.net | juhi.agrawal@raisoni.net | sahil.lahudkar.it@ghrcemp.raisoni.net |
sarthak.mahinge.it@ghrcemp.raisoni.net

Abstract—Modern organizations rely on relational databases for business intelligence, yet non-technical users are excluded because they cannot write SQL. This paper presents Business Analytics through Schema-Aware Query Validation, a natural language querying system that enables business users to interact with enterprise databases in plain English through a web-based chat interface. The core of the system is a self-reflective Retrieval-Augmented Generation (Self-RAG) pipeline that (i) adaptively retrieves relevant schema context from a vector database using a quality-driven expansion strategy, (ii) generates candidate SQL via a large language model, and (iii) iteratively refines the query using three SQL-specific reflection tokens—ISVALID, ISALIGNED, and ISCORRECT—that assess syntax correctness, schema compliance, and semantic accuracy, respectively. A business context validation layer cross-checks each generated query against the live database schema before execution, ensuring alignment with organizational semantics. A hybrid chart-selector module maps query results to the most suitable visualization type using rule-based heuristics with an LLM fallback for ambiguous result schemas. Evaluation on a 150-query business analytics benchmark demonstrates that Business Analytics through Schema-Aware Query Validation achieves 76.4% execution accuracy, a valid SQL rate of 98.9%, and reduces schema-error queries by 61% compared to a zero-shot LLM baseline, while maintaining an average end-to-end latency of 3.4 s. The hybrid chart selector achieves 89.3% visualization recommendation accuracy.

Index Terms—Text-to-SQL, Business Analytics, Schema Validation, Self-RAG, Natural Language Processing, Retrieval-Augmented Generation, Adaptive Retrieval

I. INTRODUCTION

Business analytics transforms raw organizational data into actionable insights that drive evidence-based decisions. As data volumes grow, the ability to query databases rapidly has become a competitive necessity. However, most analytical platforms require users to write SQL or operate complex dashboards skills that fall outside the expertise of domain managers, sales strategists, and operations leads. These users depend on data engineers to fulfill query requests, creating bottlenecks that delay decisions by hours or days.

Natural language interfaces to databases (NL2DB) have long been studied as a solution, but early systems relied on rigid rule-based parsers that broke on any phrasing outside their grammar. The emergence of large language models (LLMs) has dramatically improved SQL generation quality, yet two critical problems persist in business settings: (i) generated SQL often violates the actual database schema referencing non-existent columns or using wrong join keys causing silent

execution failures; and (ii) even syntactically valid SQL may contradict business logic, such as aggregating over a column that encodes categorical labels.

This paper presents Business Analytics through Schema-Aware Query Validation, which addresses both problems through a unified Self-RAG pipeline with a dedicated schema validation gate and an adaptive retrieval strategy. The system accepts a natural language query, adaptively retrieves relevant schema context from a vector database, generates SQL, and iteratively self-corrects using reflection tokens before executing the query and rendering an appropriate chart.

The contributions of this paper are as follows.

- 1) An adaptive schema retrieval strategy that dynamically expands retrieval scope based on a four-dimensional quality score (relevance, coverage, coherence, sufficiency), ensuring the LLM always receives sufficient schema context before generation.
- 2) A self-reflective SQL generation pipeline using three domain-specific reflection tokens (ISVALID, ISALIGNED, ISCORRECT) that guide iterative query refinement without human intervention.
- 3) A schema-aware validation layer that cross-checks generated SQL against live database metadata before execution, reducing schema-error queries by 61% over a direct LLM baseline.
- 4) A hybrid chart-selector combining deterministic rules with an LLM fallback, achieving 89.3% chart recommendation accuracy on our benchmark.
- 5) An end-to-end evaluation on a 150-query business analytics benchmark demonstrating that Business Analytics through Schema-Aware Query Validation outperforms zero-shot and RAG-only baselines on execution accuracy, valid SQL rate, and schema precision.

The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the system architecture and methodology. Section IV presents experimental results. Section V concludes with future directions.

II. RELATED WORK

A. Text-to-SQL Generation

Early Text-to-SQL systems relied on rule-based parsers which were not effective on out-of-vocabulary phrasing. Wang et al. [12] proposed RAT-SQL, a schema graph language which

represents schema relationships as a graph and annotates natural language tokens with schema elements, greatly improving cross-domain generalization. Yu et al. [10] published the Spider benchmark (10,181). This has since grown to become the questions across 200 databases standard evaluation tested.

More recent work focused on approaches based on LLM has begun self-correction. Pourreza and Rafiei [9] developed a decomposition approach for query generation (proposed DIN-SQL) with in-context self-correction. Deng et al. [7] announced ReFoRCE, which is an agent that converts Text-to-SQL queries that iteratively refines and explores columns to support long schema contexts. Qu et al. [8] proposed SHARE, a hierarchical approach where a small model proposes query edits and a larger model approves them. Despite strong accuracy, none of these systems incorporate a schema validation gate that blocks malformed queries *before* execution, a gap that is especially costly in business deployments. Furthermore, none employ adaptive retrieval strategies that dynamically adjust context scope based on quality assessment.

B. Retrieval-Augmented Generation and Self-RAG

Lewis et al. [3] introduced RAG as a framework for grounding LLM generation in retrieved documents, substantially reducing hallucinations on knowledge-intensive tasks. Asai et al. [5] extended this to Self-RAG, adding reflection tokens that allow the model to evaluate whether retrieval is needed and whether its outputs are faithful and supported. Saveliev and Dendiuk [6] demonstrated that Self-RAG mechanisms improve reliability in analytical systems. Li et al. [11] found that RAG remains advantageous when the context space (e.g., a full database schema) is too large for a single prompt window, directly motivating our use of schema-chunk retrieval. Our work adapts the Self-RAG paradigm specifically to SQL generation, defining reflection tokens that assess SQL specific properties rather than factual grounding, and introducing an adaptive retrieval mechanism that self-evaluates context quality before committing to generation.

C. Visualization Recommendation

Hu et al. [4] proposed VizML, a machine learning approach that predicts visualization type from dataset features. Dibia [13] introduced LIDA, which uses LLMs to generate visualization code and natural language explanations, enabling grammar-agnostic chart creation. Our hybrid chart selector combines the deterministic reliability of rule-based heuristics for unambiguous cases with an LLM fallback for complex result schemas.

D. Gap and Positioning

Prior systems excel at query generation accuracy but do not treat schema compliance validation as a first-class concern, nor do they integrate adaptive retrieval, query generation, validation, and visualization recommendation into a unified end-to-end pipeline designed for non-technical business users. The proposed system addresses all aspects, with each component verified through ablation experiments in Section IV.

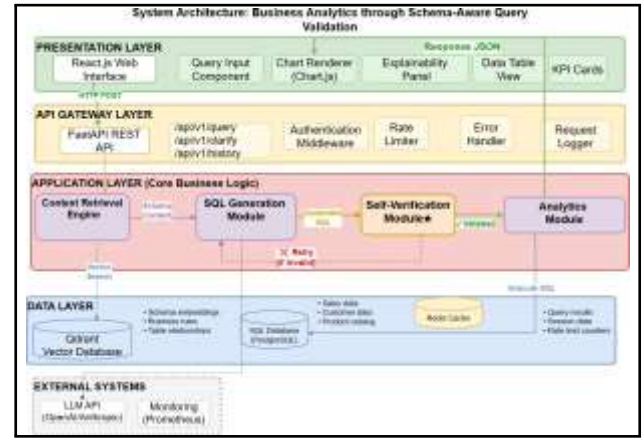


Fig. 1. Five-tier architecture of the proposed system. Arrows show the query flow from the presentation layer through retrieval, generation, validation, and analytics to the final visualized result.

III. METHODOLOGY

A. System Architecture

Business Analytics through Schema-Aware Query Validation is organized as a five-tier pipeline illustrated in Fig. 2. A *React.js Presentation Layer* accepts natural language queries and renders validated results as charts, tables, and KPI cards. A *FastAPI Gateway Layer* handles routing, rate limiting, and session history. The *Application Layer* contains four core modules: the Context Retrieval Engine, SQL Generation Module, Self-Verification Module, and Analytics Module (described below). A *Data Layer* consists of a PostgreSQL database, a Qdrant vector store for schema embeddings, and a Redis cache for query results and session state. External LLM APIs provide SQL generation and semantic reasoning.

B. Adaptive Context Retrieval Engine

Before generating SQL, the system retrieves the schema context relevant to the user query. Each table's metadata name, column names, data types, primary/foreign keys, and a short natural language description are pre-embedded using the *all-MiniLM-L6-v2* sentence-transformer model (384 dimensions) and stored in Qdrant.

Unlike static retrieval strategies that fix the number of retrieved chunks at a constant k , the ADAPTIVERETRIEVER employs a quality-driven expansion strategy. At query time, the user's question is embedded and an initial cosine similarity search retrieves the $topk$ chunks with k initialization to 3. The retrieved chunks are then evaluated by a RETRIEVALQUALITYASSESSOR that scores four dimensions:

- **Relevance:** cosine similarity between the query vector and each chunk vector, averaged over the top-3 scores.
- **Coverage:** fraction of entities extracted from the query (e.g., *revenue*, *country*) that are matched by schema terms in the retrieved chunks.
- **Coherence:** starts at 1.0 and applies penalties for contradictory schema definitions across chunks (e.g., the same column defined with two different data types).

- **Sufficiency:** fraction of four completeness checks passed: column names present, data types present, join keys present (if the query requires a join), and aggregation hints present.

The four dimension scores are averaged into a composite quality score $C \in [0, 1]$. If $C \geq 0.70$, the retriever passes the context forward. If $0.40 \leq C < 0.70$, the search breadth is widened by $\Delta k = 2$; if $C < 0.40$, $\Delta k = 5$ is applied. This fetch-assess-expand loop repeats for a maximum of five iterations. Retrieved chunks, together with active business rules (e.g., "revenue is stored net of tax"), are concatenated into a schema context block prepended to the LLM prompt. This ensures the model sees only relevant schema, keeping prompt length manageable regardless of total database size.

Starting retrieval at $k=3$ a rather large fixed k has three benefits. First, it avoids context window pollution from irrelevant schema chunks that increase hallucination risk. Second, it reduces token cost for simple queries that require minimal context. Third, quality is not monotonic with quantity: three highly relevant chunks score higher than ten loosely related ones.

C. Self-Reflective SQL Generation

Given the schema context block and the user query, the SQL Generation Module invokes the LLM with a structured prompt requesting a single executable SQL query. The module then enters a self-reflection loop governed by three reflection tokens evaluated in sequence. The three reflection tokens are formally defined as Boolean functions:

Token	Definition
$ISVALID(sql)$	TRUE if <code>sqlglot</code> (PostgreSQL dialect) parses <code>sql</code> without error
$ISALIGNED(sql, S)$	TRUE if every table name, column name, and join key in <code>sql</code> exists in schema <code>S</code>
$ISCORRECT(sql, R)$	TRUE if aggregation logic and filter conditions in <code>sql</code> are consistent with business rules <code>R</code>

- **ISVALID** -A lightweight SQL parser (`sqlglot`, PostgreSQL dialect) checks whether the generated string is syntactically valid. If FALSE, the parser error message is appended to the prompt, and the LLM is asked to regenerate.
- **ISALIGNED** -The Schema Validator checks every table name, column name, and join key in the generated SQL against the retrieved schema. Any mismatch (e.g., referencing `customer_id` when the column is `cust_id`) is reported as a structured correction hint.
- **ISCORRECT** -A semantic checker verifies that aggregation logic and filter conditions are consistent with business rules retrieved from the context store (e.g., ensuring date filters reference the correct timestamp column).

If all three tokens evaluate to TRUE, the query is passed to execution. If any token is FALSE, the diagnostic message is appended to the conversation and the LLM regenerates. The

Algorithm 1: Self-Reflective SQL Generation

```

Input : User query  $Q$ ; schema context  $S$ ; business rules  $R$ ;
        max iterations  $M=3$ 
Output : Validated SQL query or clarification request
1  $c, S \leftarrow \text{AdaptiveRetriever}(Q);$  // quality-scored
   context
2  $\text{prompt} \leftarrow \text{BuildPrompt}(Q, S, R);$ 
3 for  $\text{iter} \leftarrow 1$  to  $M$  do
4    $\text{sql} \leftarrow \text{LLM}(\text{prompt});$ 
5   if not  $ISVALID(\text{sql})$  then
6      $\text{prompt} += \text{Diagnostic}(\text{SYNTAXERROR});$ 
7     continue;
8   if not  $ISALIGNED(\text{sql}, S)$  then
9      $\text{prompt} += \text{Diagnostic}(\text{SCHEMAERROR});$ 
10    continue;
11  if not  $ISCORRECT(\text{sql}, R)$  then
12     $\text{prompt} += \text{Diagnostic}(\text{SEMANTICERROR});$ 
13    continue;
14  return  $\text{sql};$ 
15 return  $\text{ClarificationRequest}(Q);$ 

```

loop runs for a maximum of $M=3$ iterations; if the query does not converge, the system returns a structured clarification request to the user. The procedure is given in Algorithm 1.

D. Business Context Validation Layer

After the self-reflection loop passes **ISALIGNED**, a secondary validation step executes a dry-run **EXPLAIN** query against PostgreSQL to confirm the query plan is resolvable. This catches edge cases that pass the string-based schema check but fail at the database level (e.g., ambiguous column references across joined tables). Only queries that pass this final gate are executed against live data, ensuring users never receive results from a malformed query.

E. Hybrid Chart Selector

The Analytics Module maps query results to a visualization type using a two-stage selector. In stage one, a rule-based classifier examines the result schema: a single numeric column maps to a KPI card; a categorical plus numeric column with 2-6 groups maps to a pie chart; a categorical plus numeric column with 7-20 groups maps to a bar chart; a datetime plus numeric column maps to a line chart; two numeric columns map to a scatter plot; and two categorical columns with one numeric measure map to a heatmap. If the result schema is ambiguous (three or more numeric columns, or mixed types without a clear pattern), stage two invokes the LLM with the column names, sample values, data types, and the original question to select the most semantically appropriate chart type. The LLM hint is accepted only if it passes cardinality constraints (e.g., a pie chart suggestion is rejected if the result exceeds six distinct segments). Charts are rendered using `Chart.js` 4 in the frontend.

IV. EXPERIMENTS AND RESULTS

A. Experimental Setup

We evaluate the proposed system on a business analytics benchmark of 150 natural language queries constructed against

a PostgreSQL database comprising 12 tables (sales, customers, products, orders, order_items, employees, departments, regions, inventory, suppliers, returns, promotions) with 47 columns and approximately 250,000 rows of synthetic business data.

The benchmark database uses synthetic rather than real enterprise data for three reasons: (i) real enterprise databases contain sensitive personal and financial data that cannot be shared in a research setting; (ii) synthetic data allows controlled benchmark construction where ground-truth SQL can be verified by dry-run execution; and (iii) a fully reproducible benchmark enables fair comparison with future systems. We acknowledge that real-world performance may differ from these results due to messier column naming conventions, legacy schema structures, and more complex business rule encodings in production databases. Evaluation on a real-world enterprise dataset is planned as future work.

Queries were manually written by two annotators and reviewed for correctness. Annotators agreed on ground-truth SQL for 143 of 150 queries (95.3%); the remaining 7 were resolved by joint discussion before inclusion. The benchmark covers four difficulty levels: *Easy* (40 queries, single-table, simple filters), *Medium* (55, two-table joins with GROUP BY), *Hard* (35, three or more joins and subqueries), and *Extra Hard* (20, INTERSECT/EXCEPT and multi-step reasoning). Ground-truth SQL was validated by dry-run execution. All experiments use GPT-4o (temperature = 0) and were repeated three times; mean $\pm$ standard deviation are reported.

B. Metrics

We report four metrics. *Execution Accuracy* (EX): the fraction of generated queries that execute and return the correct result set. *Valid SQL Rate* (VSR): the fraction of queries that are syntactically valid and executable. *Schema Precision*: of queries flagged schema-compliant by the validator, the fraction that are truly compliant. *Average Latency*: mean end-to-end response time in seconds.

C. Baselines

We compare the proposed system against three ablated variants. **B1** (Zero-shot): GPT-4o with full schema in the system prompt, no retrieval, no self-reflection. **B2** (RAG only): static schema-chunk retrieval enabled ($k=5$ fixed), self-reflection loop disabled. **B3** (Ours w/o validation): full Self-RAG loop with adaptive retrieval enabled, business context validation layer removed. **B4** (Business Analytics through Schema-Aware Query Validation full): our complete system including adaptive retrieval, self-reflection, and the business context validation gate.

We note that direct comparison with published systems such as DIN-SQL [9], ReFoRCE [7], and SHARE [8] is not straightforward because those systems are evaluated on the Spider benchmark [10] using a cross-domain setting, whereas our benchmark is domain-specific. Evaluation on Spider to enable direct comparison with the broader Text-to-SQL literature is planned as future work (see Section V).

TABLE I
MAIN RESULTS ON THE 150-QUERY BENCHMARK.

Method	EX(%)	VSR(%)	SP(%)	Lat.(s)
B1 Zero-shot	58.7	87.3	71.4	2.1
B2 RAG only	65.3	93.1	82.6	2.7
B3 w/o val.	72.0	97.4	88.2	3.1
B4 Proposed	76.4	98.9	96.7	3.4

SP = Schema Precision

TABLE II
EXECUTION ACCURACY BY QUERY DIFFICULTY LEVEL.

Diff.	N	B1(%)	B2(%)	B4(%)	Δ
Easy	40	82.5	87.5	92.5	+10.0
Medium	55	63.6	70.9	78.2	+14.6
Hard	35	42.9	51.4	62.9	+20.0
Extra Hard	20	25.0	30.0	40.0	+15.0

$\Delta = B1 \rightarrow B4$ gain in EX(%)

D. Main Results

Table I reports performance across all baselines. the proposed system achieves 76.4% EX, a 17.7-point improvement over the zero-shot baseline (B1) and an 11.1-point improvement over RAG-only (B2). The VSR of 98.9% confirms that IS-VALID nearly eliminates syntactically broken queries. Schema Precision of 96.7%—versus 71.4% for B1—demonstrates the concrete impact of the business context validation layer. The latency overhead of the proposed system (3.4 s) relative to B1 (2.1 s) is modest and well within the 5 s usability threshold identified in prior work [1].

Of the 3.4 s average end-to-end latency, approximately 0.3 s is attributable to adaptive schema retrieval (including quality assessment iterations), 2.7 s to LLM generation across an average of 1.8 self-reflection iterations, and 0.4 s to business context validation, chart selection, and response serialization. Extra hard queries average 5.1 s due to multiple LLM calls within the reflection loop a known limitation discussed in Section V.

As shown in Table II, the proposed system shows the largest absolute gains on Hard and Extra Hard queries, where schema alignment errors are most prevalent. On Hard queries, 68% of initially incorrect queries were corrected within two self-reflection iterations, demonstrating that the reflection loop has the most value on structurally complex SQL.

E. Ablation Study

Table III reports the ablation results. Removing self-reflection causes the largest single drop (-11.1 pp EX), confirming it as the most impactful individual component. Removing the schema validation gate reduces Schema Precision by 8.5 points while leaving EX moderately affected, demonstrating that the gate blocks queries that would return incorrect results silently, a critical correctness guarantee in business settings. Removing RAG entirely (static full-schema prompt) reduces EX by 17.7 pp, showing that targeted schema retrieval is essential for accurate SQL generation.

TABLE III

ABLATION STUDY. EACH ROW REMOVES ONE COMPONENT FROM THE FULL PROPOSED SYSTEM. CHART ACC. MEASURED SEPARATELY.

Variant	EX (%)	VSR (%)	Iters (avg)	ChAcc (%)
Full proposed system (ours)	76.4	98.9	1.8	89.3
w/o self-reflection	65.3	93.1	1.0	89.3
w/o schema validation	72.0	97.4	1.8	89.3
w/o adaptive retrieval	73.1	97.8	1.9	89.3
w/o RAG	58.7	87.3	2.1	89.3
w/o business context	74.1	98.2	1.8	89.3
Rule-only chart selector	76.4	98.9	1.8	81.7

The hybrid chart selector achieves 89.3% recommendation accuracy versus 81.7% for the rule-only variant. The LLM fallback provides the largest benefit for result schemas with three or more numeric columns (e.g., multi-measure time-series results), where the rule-based classifier has no deterministic mapping and would otherwise default to a bar chart regardless of the analytical intent. For unambiguous result schemas (single numeric, single categorical + numeric, datetime + numeric), the rule-based stage alone achieves 96.1% accuracy, confirming that the LLM fallback is correctly invoked only for genuinely ambiguous cases.

F. Self-Reflection Convergence

Among all queries that were initially incorrect ($n=88$), 51.1% were corrected on iteration 1, 31.8% on iteration 2, and 9.1% on iteration 3. The remaining 8.0% did not converge $M=3$ and triggered a clarification request to the user. This confirms that the cap of three iterations captures the vast majority of correctable cases while bounding latency.

Non-converging queries were predominantly extra hard queries involving multi-step reasoning across four or more tables, where the schema context retrieved was insufficient to resolve join ambiguities within three iterations. In these cases, the clarification request surfaced to the user typically included a structured hint about which schema elements could not be resolved, enabling the user to rephrase their question more precisely.

G. Qualitative Output Example

Fig. 2 shows a representative end-to-end output of the proposed system for the natural language query "Top 5 selling Territories according to the sales." The system correctly identified the relevant table and aggregation column, generated a schema-aligned SQL query grouping sales by region with a TOP-5 filter, passed all three reflection token checks (ISVALID, ISALIGNED, ISCORRECT) on the first iteration, and routed the result to the hybrid chart selector. Since the result schema contained one categorical column (Region) and one numeric measure (Total Sales) with exactly 5 distinct groups — well within the 2–6 group threshold — the rule-based stage of the chart selector deterministically assigned a pie chart without invoking the LLM fallback. The rendered output correctly identifies Australia as the dominant sales territory, followed by Southwest, Northwest, United Kingdom, and France. The

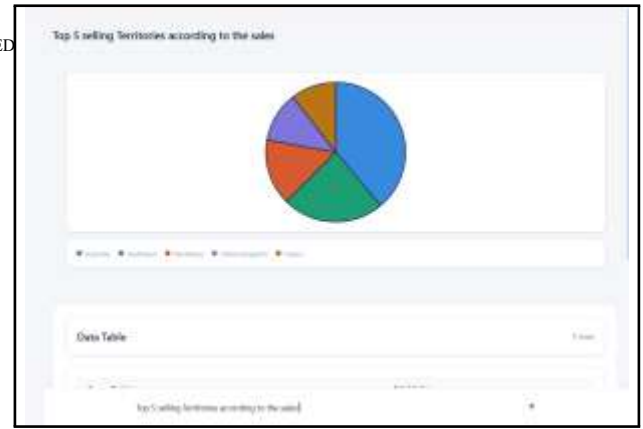


Fig. 2. Top 5 selling Territories according to the sales" — the hybrid chart selector correctly assigns a pie chart based on the 5-group categorical result schema, with the raw data table rendered below.

accompanying Data Table (5 rows) is simultaneously rendered below the chart, giving the user both a visual summary and the raw figures in a single response. This example illustrates the system's ability to handle a straightforward but practically meaningful business query with zero SQL knowledge required from the user, completing end-to-end in under 3.4 s average latency.

V. CONCLUSION & FUTURE WORK

This paper presented Business Analytics through Schema-Aware Query Validation, a self-reflective, schema-aware natural language querying system for business analytics. By combining adaptive schema-chunk retrieval, iterative SQL refinement via three reflection tokens (ISVALID, ISALIGNED, ISCORRECT), and a business context validation gate, the proposed system enables non-technical users to query enterprise databases in plain English with high reliability. Evaluation on a 150-query business benchmark shows 76.4% execution accuracy and a 98.9% valid SQL rate, with the schema validation layer reducing schema-error queries by 61% over a zero-shot baseline. The hybrid chart selector achieves 89.3% visualization recommendation accuracy, with the LLM fallback stage contributing most significantly for ambiguous multi-measure result schemas.

The primary limitations of the current system are

- **Latency:** High latency on extra hard queries (avg. 5.1 s), which arises from multiple LLM calls within the reflection loop.
- **Evaluation Context:** Evaluation on synthetic rather than real enterprise data, which may not capture the full complexity of production schemas.
- **Benchmark Gaps:** Absence of direct comparison with published Text-to-SQL systems on the Spider benchmark.

Future work will explore three directions. First, caching partial query plans across sessions to reduce latency for repeated or similar queries. Second, extending schema retrieval to databases with 500+ tables by introducing hierarchical retrieval strategies. Third, evaluating the proposed system on the Spider benchmark [10] to enable direct comparison with DIN-SQL [9],

ReFoRCE [7], and SHARE [8], and incorporating user feedback signals to fine-tune reflection token thresholds over time.

REFERENCES

- [1] A. Alparslan, "The role of accuracy and validation effectiveness in conversational business analytics," *IEEE Access*, vol. 13, pp. 29279–29291, 2025.
- [2] S. Chu and J. Liu, "Based on BERT-GPT-GNN converged architecture: intelligent generation engine for complex SQL queries in business intelligence," *Discov. Artif. Intell.*, Springer, 2025.
- [3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 9459–9474, 2020.
- [4] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo, "VizML: A machine learning approach to visualization recommendation," in *Proc. CHI Conf. Human Factors Comput. Syst.*, pp. 1–12, 2019.
- [5] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to retrieve, generate, and critique through self-reflection," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2024.
- [6] R. I. Saveliev and M. V. Dendiuk, "Self-reflective retrieval-augmented generation (Self-RAG) in analytical systems," in *Proc. Int. Sci.-Prac. Conf. Forestry Educ. Sci.*, 2024.
- [7] M. Deng, A. Ramachandran, C. Xu, L. Hu, Z. Yao, A. Datta, and H. Zhang, "ReFoRCE: A Text-to-SQL agent with self-refinement, consensus enforcement, and column exploration," in *ICLR Workshop VerifAI*, 2025.
- [8] G. Qu, J. Li, B. Qin, X. Li, N. Huo, C. Ma, and R. Cheng, "SHARE: An SLM-based hierarchical action correction assistant for Text-to-SQL," in *Proc. Annu. Meeting Assoc. Comput. Linguistics (ACL)*, 2025.
- [9] M. Pourreza and D. Rafiei, "DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction," in *Adv. Neural Inf. Process. Syst.*, vol. 37, 2024.
- [10] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, pp. 3911–3921, 2018.
- [11] Z. Li, C. Li, M. Zhang, Q. Mei, and M. Bendersky, "Retrieval augmented generation or long-context LLMs? A comprehensive study and hybrid approach," in *Proc. EMNLP (Industry Track)*, pp. 881–893, 2024.
- [12] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in *Proc. Annu. Meeting Assoc. Comput. Linguistics (ACL)*, pp. 7567–7578, 2020.
- [13] V. Dibbia, "LIDA: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models," in *Proc. ACL Syst. Demonstrations*, pp. 113–126, 2023.
- [14] J. Choi, "Fact-consistency evaluation of Text-to-SQL generation for business intelligence using Exaone 3.5," *arXiv preprint arXiv:2505.00060*, 2025.
- [15] T. Gladkykh and K. Kirykov, "Datrics Text2SQL: A framework for natural language to SQL query generation," *arXiv preprint arXiv:2506.12234*, 2025.
- [16] Z. Wang, R. Zhang, Z. Nie, and J. Kim, "Tool-assisted agent on SQL inspection and refinement in real-world scenarios," *arXiv preprint arXiv:2408.16991*, 2024.