Two-Layer RAG for Value Grounding and Schema Linking in LLM-Based Text-to-SQL Systems

João Victor Monteiro Macedo[1], **Renato Miyaji**[1], **Renato Moulin**[1], **Leonardo Machado**[1]

[1]Grupo Visagio

{joao.macedo,renato.miyaji,renato.moulin,leonardo.machado}@visagio.com

Abstract. *Large Language Models translate natural language into SQL. In environments with dynamic schemas, models hallucinate column names and filter values absent from the target database. We present a two-layer Retrieval-Augmented Generation architecture that addresses schema linking and value grounding within an orchestration pipeline. The structural layer retrieves validated SQL examples via cosine similarity over dense embeddings, while the semantic layer extracts entities from the user query and matches them against real column values through a MinHash-then-Embedding cascade. The architecture improved Execution Accuracy from 23.6% to 42.5%. By successfully grounding filter values, the semantic layer reduced the Empty Result Rate from 26.4% to 7.5%.*

1. Introduction

Text-to-SQL systems translate natural language questions into executable SQL. LLM-based approaches achieved good results on benchmarks such as Spider [Yu et al. 2018] and BIRD [Li et al. 2024]. However, deployment in corporate environments exposes failure modes that benchmarks do not capture [Floratou et al. 2024]. Two problems cause execution failures. First, schema linking requires mapping entities mentioned by the user to the correct tables and columns in the database. Second, value grounding requires literal values used in WHERE clauses to correspond to data stored in the target columns, rather than variants generated by the model. Given the query "What is the revenue for the São Paulo branch last quarter?", an ungrounded model may produce WHERE branch = 'Sao Paulo' when the stored value is 'SP - São Paulo', yielding an empty result set without raising an execution error.

This paper presents a two-layer RAG architecture designed to address both problems. This architecture is implemented as an enhancement to a pre-existing agentic Text-to-SQL system. The architecture combines (1) a structural layer that retrieves historically validated SQL queries via cosine similarity over dense embeddings, and (2) a semantic layer that performs entity extraction followed by a MinHash-then-Embedding cascade to match user-mentioned values against column contents. Both layers are coordinated by a Dynamic Context Orchestrator, which parallelizes context loading to control latency overhead.

To evaluate the proposed architecture, we conducted a real-world case study at a consulting firm that operates across diverse sectors, including Financial Services, Energy, and Manufacturing. Consequently, our experiments utilize proprietary client datasets, demonstrating the system's robustness and adaptability in complex, domain-specific corporate environments.

2. Related Work

LLM-based Text-to-SQL research evaluates GPT models [Rajkumar et al. 2022] and benchmarks LLM capabilities [Gao et al. 2024]. DIN-SQL [Pourreza and Rafiei 2024] decomposes the problem into schema linking, classification, and generation sub-tasks. C3 [Dong et al. 2023] applies ChatGPT in a zero-shot setting with calibrated prompting. Schema linking is a bottleneck [Lei et al. 2020]. Existing approaches rely on lexical matching or trained classifiers, but these fail in environments where schemas are dynamic (e.g., when tables are created on-the-fly from user-uploaded spreadsheets). Floratou et al. [Floratou et al. 2024] note that NL2SQL systems face challenges distinct from benchmark settings, including domain-specific vocabularies and evolving schemas. RAG is applied to QA tasks, but its use for value grounding in Text-to-SQL is underexplored. Few-shot example retrieval is established [Brown et al. 2020], but the combination of structural retrieval (SQL examples) with semantic retrieval (database values) into a pipeline has not been addressed. MinHash [Broder 1997], used for near-duplicate detection, has not been applied as a first-stage filter in Text-to-SQL value grounding.

3. Proposed Method

3.1. System Overview

The base architecture upon which this study is conducted is an LLM-based conversational agent designed for enterprise data analysis, built on the Pydantic AI framework [Pydantic AI Team 2024]. It receives natural language queries and translates them into SQL executable over a PostgreSQL database containing heterogeneous data uploaded by users. Figure 1 illustrates the architecture.

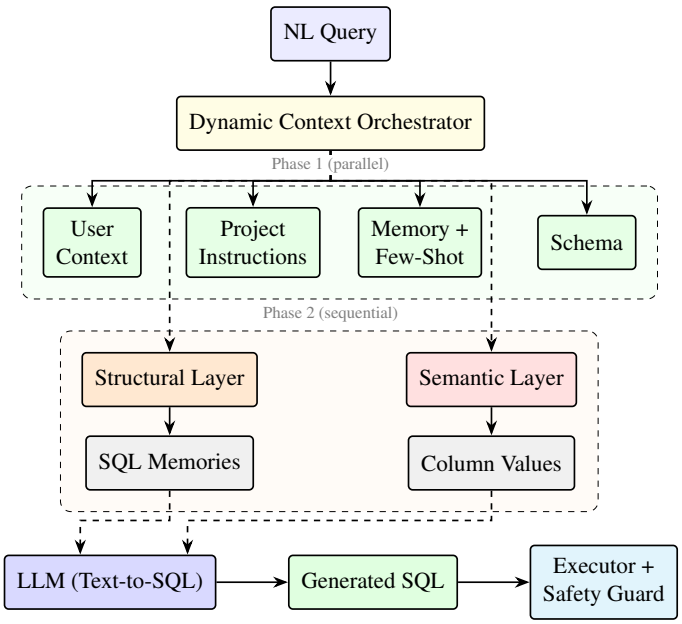

Figure 1. Architecture of the Dynamic Context Orchestrator.

The context retrieval and injection pipeline is governed by the Dynamic Context Orchestrator, which operates in a two-phase execution model designed to balance context aggregation with low-latency constraints. Phase 1 implements a parallelized retrieval protocol. It concurrently initializes four independent context loaders—each maintaining an

isolated database session to mitigate I/O bottlenecks. These loaders extract fundamental metadata: (a) user profile information to establish session context, (b) domain-specific project business rules to constrain the generation space, (c) historical behavioral memories paired with relevant few-shot exemplars, and (d) the structural schema encompassing permitted tables and detailed column semantics.

Subsequently, Phase 2 is executed sequentially, as it explicitly depends on the state caches and semantic bounds materialized during Phase 1. In this phase, the orchestrator triggers the dual-layer RAG architecture. As depicted in Figure 1, this involves a Structural Layer responsible for retrieving historically validated SQL structures (SQL Memories), and a Semantic Layer dedicated to grounding user-provided literal values against actual database contents (Column Values). Finally, the synthesized context from both phases is injected into the LLM Text-to-SQL pipeline. The resulting generated SQL is then routed through an execution module equipped with a safety guard, ensuring that structural and security constraints are rigorously validated prior to database execution.

3.2. Structural Layer: SQL Example Retrieval

The structural layer is engineered to resolve schema linking ambiguities. By dynamically retrieving structurally analogous and validated SQL formulations, it provides the LLM with explicit contextual demonstrations of correct table mappings and join operations relevant to the user's intent. The architecture maintains a dynamic repository of *text2sql* execution traces. Each trace encapsulates a prior natural language query paired with its corresponding SQL statement, which has been explicitly verified or corrected via closed-loop user feedback. The formal mechanism for this structural retrieval is detailed in Algorithm 1.

Algorithm 1 Structural Context Retrieval

Require: Incoming natural language query q, memory repository $\mathcal{M}$, embedding model $\text{Embed}(\cdot)$, similarity threshold $\tau_s = 0.7$, selection size $k = 3$
Ensure: Top-k structurally relevant text2sql traces

$\mathbf{e}_q \leftarrow \text{Embed}(q)$ ▷ Dynamic encoding of the input query
$\mathcal{C} \leftarrow \emptyset$ ▷ Initialize candidate set
for each trace $i \in \mathcal{M}$ **do**
  $\mathbf{e}_i \leftarrow \text{Embed}(\text{SQL}_i)$ ▷ Target encoding at request time
  $\text{sim}(q, i) \leftarrow \frac{\mathbf{e}_q \cdot \mathbf{e}_i}{\|\mathbf{e}_q\| \cdot \|\mathbf{e}_i\|}$ ▷ Compute cosine similarity
  if $\text{sim}(q, i) > \tau_s$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(i, \text{sim}(q, i))\}$ ▷ Apply threshold filtering
  end if
end for
Sort $\mathcal{C}$ in descending order based on $\text{sim}(q, i)$
return the top-k traces from $\mathcal{C}$

The retrieved context incorporates both positive demonstrations and negative constraints. Traces associated with negative user feedback are explicitly annotated as invalid, accompanied by the rationale for rejection. This contrastive prompting strategy explicitly steers the LLM away from established failure manifolds, representing a significant di-

vergence from conventional few-shot prompting paradigms [Brown et al. 2020] that rely exclusively on positive reinforcement.

3.3. Semantic Layer: Value Grounding via MinHash+Embedding Cascade

The semantic layer targets value hallucination directly through a four-stage pipeline (Figure 2).

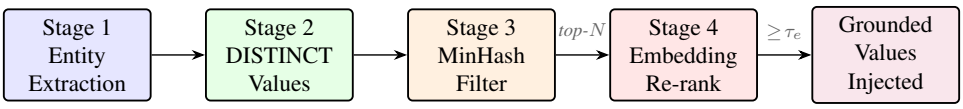

Figure 2. Four-stage pipeline of the semantic layer. MinHash acts as a filter, reducing the candidate space before embedding-based re-ranking.

Stage 1 (Entity Extraction). An LLM call extracts entities from the user query, categorizing them into four types: (a) business entities (company names, products, categories, departments); (b) specific column values likely present in the database (codes, identifiers, status labels); (c) date references (months, years, quarters, normalized as strings); and (d) numeric qualifiers (thresholds, ranges, limits).

Stage 2 (Distinct Value Collection). For each RAG-enabled text column (configured per-table and per-column), the system executes SELECT DISTINCT queries limited to $V_{\max}$ values. This configuration prevents overhead on high-cardinality columns.

Stage 3 (MinHash/Jaccard Filtering). Each extracted entity is compared against collected values using Jaccard similarity estimated via MinHash [Broder 1997], built from character 2-shingles with $n_{\text{perm}} = 128$ permutations. As a fast path, a substring match with Unicode normalization (accent removal and case-folding) is applied first; values matched by substring receive similarity 1.0 and bypass MinHash. Remaining values are scored by MinHash Jaccard; those exceeding $\tau_m = 0.15$ are retained as candidates. The threshold is intentionally low to favor recall, as the re-ranking stage (Stage 4) provides precision.

Stage 4 (Embedding Re-ranking). The top-N candidates from Stage 3 ($N = 30$) are re-ranked using cosine similarity between dense embeddings of the entity and each candidate value. Only values with sim $\geq \tau_e = 0.2$ are injected into the prompt as grounded values, up to a maximum of 10 per entity.

MinHash operates in $O(n)$, reducing distinct values to a candidate set before the embedding computation is applied. The grounded values are injected into the prompt as structured context blocks with instructions to use them in filters.

3.4. Feedback-Driven Continuous Learning

Both layers utilize a feedback loop. When a user approves or corrects a response, the system (i) extracts entities and generates embeddings for the query-SQL pair; (ii) persists it as a project memory with feedback polarity and correction text; and (iii) makes it available for structural retrieval. Negative examples are annotated with the error reason, functioning as counter-factual demonstrations (analogous to instance-level RLHF [Ouyang et al. 2022]) that reduce recurrence of corrected errors.

3.5. Experimental Details

The experiments were conducted leveraging Gemini-3-Flash for the main agent and for specific tools. The Text-to-SQL model was integrated with a self-repair loop that permits up to three automatic retries upon execution failure. The system was evaluated against a private database from the offshore oil and gas industry, simulating the complexity of real-world fleet operations. The dataset consists of 106 natural language queries mapped to gold SQL across three interconnected domains: daily production and downtime metrics, maintenance work orders with categorical statuses, and supply chain logistics involving delivery delays. This benchmark captures domain-specific challenges such as non-obvious business rules (e.g., technical definitions of "deepwater") and literal value mismatches (e.g., vessel names with complex diacritics). System efficacy was assessed via four interlinked metrics. Execution Accuracy (EX) measures exact result-set matches against gold SQL. Value Matching Rate (VMR) computes the proportion of correctly grounded literal values. Empty Result Rate (ERR) tracks queries falsely returning zero rows, explicitly excluding legitimately empty gold results to prevent skewed evaluation. These metrics operate causally: higher VMR reduces literal hallucinations, thereby lowering ERR, which directly drives overall EX improvements. Finally, median latency (p50) evaluates the computational tradeoff of this retrieval cascade.

4. Evaluation and Preliminary Results

Table 1 presents the quantitative results comparing a zero-shot baseline (schema-only) against the proposed two-layer RAG pipeline. The two-layer RAG architecture improved EX from 23.6% to 42.5%, an absolute gain of 18.9 percentage points. While this absolute accuracy may appear modest compared to highly curated benchmarks, it reflects the inherent complexity, noise, and ambiguity of the real-world corporate datasets evaluated in this study. The semantic layer successfully grounded 83% of the required filter values (VMR), which directly contributed to reducing the ERR from 26.4% to 7.5%. This reduction confirms that a significant portion of baseline failures stems from value hallucinations rather than logic errors.

Table 1. Ablation study on the two-layer RAG pipeline ($N = 106$ queries per condition).

Condition	EX (%)	VMR (%)	ERR (%)	Latency p50 (ms)
No RAG (Baseline)	23.6	0.0	26.4	3152
Two-Layer RAG (Full)	**42.5**	83.0	7.5	8518

The improvement is statistically significant ($p < 0.001$ per McNemar's test), indicating that the RAG layers provide essential context that the model cannot infer from the schema alone. While the median latency increased by approximately 5 seconds due to the MinHash-then-Embedding cascade and few-shot retrieval, the tradeoff is justified by the substantial increase in system reliability and the elimination of "false empty" responses that undermine user trust.

4.1. Error Mitigation Analysis

The architecture resolved several failure modes identified in the baseline. Beyond the quantitative gains, we cataloged common errors such as empty results due to partial names

and schema aliasing. Table 2 details how the distinct RAG layers mitigate these specific challenges in the offshore oil and gas context.

Table 2. Complex use cases and error mitigation strategies provided by the two-layer RAG.

Query Intent	Failure Mode (No RAG)	Correction Strategy	Active Layer
"Maintenance cost for orders waiting parts"	Model infers `status = 'waiting parts'`, causing an Empty Result.	Injects real DB value: `'WIP - Waiting Parts'` into the prompt.	**Semantic Layer** (MinHash+Embedding)
"Suppliers stuck in customs"	Model translates to English poorly: `delivery_status = 'Customs'`.	Maps Portuguese intent to the exact categorical `'Delayed in Customs'`.	**Semantic Layer** (Cross-lingual mapping)
"Most delayed projects"	Model uses `LIMIT 100` inside a `SUM(budget)` query, altering results.	Retrieves negative memory: "Do not use LIMIT with aggregate functions".	**Structural Layer** (Negative Memory)
"Suppliers delaying the Errea Wittu FPSO"	Model filters on text: `delivery_status LIKE '%delay%'`.	Retrieves positive memory: "Use `delay_days > 0` for delays".	**Structural Layer** (Few-Shot Demo)

Latency overhead is mitigated through three architectural optimizations: (i) parallel execution of Phase 1 context loaders to minimize I/O bottlenecks; (ii) stateful caching of schemas and exemplars to prevent redundant data fetching across tool invocations; and (iii) conditional pruning of the semantic pipeline for tables lacking RAG configurations. Additionally, the MinHash-then-Embedding cascade significantly curtails computationally expensive dense embedding calls. Finally, a closed-loop feedback mechanism facilitates self-correction by injecting annotated execution errors (e.g., misusing `LIMIT` on aggregates) as contrastive demonstrations. Unlike conventional few-shot prompting, this strategy establishes explicit failure manifolds, structurally steering the model away from known failure patterns.

5. Conclusion

We presented a two-layer RAG architecture for mitigating schema linking and value grounding failures in LLM-based Text-to-SQL systems. The combination of structural retrieval with semantic retrieval reduces hallucinations in corporate environments. Contributions include (i) framing value grounding as a two-phase retrieval problem (approximate filtering followed by semantic re-ranking); (ii) integrating negative feedback as counter-factual examples; and (iii) a parallel orchestration architecture that enables context retrieval with controlled latency. Future work should expand our evaluation to compare the architecture against stronger Text-to-SQL baselines. In addition, they should conduct an ablation study to isolate individual impacts of the structural layer, the semantic layer, and the injection of negative memories. Finally, they should investigate approximate nearest-neighbor indices (HNSW/IVF) and domain-specific embedding fine-tuning to further optimize system latency and recall.

References

Broder, A. Z. (1997). On the resemblance and containment of documents. *Proceedings of the Compression and Complexity of Sequences*, pages 21–29.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901.

Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Lin, J., Lou, D., et al. (2023). C3: Zero-shot text-to-SQL with ChatGPT. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Floratou, A., Psallidas, F., Agrawal, A., et al. (2024). NL2SQL is not solved yet: Challenges in real-world enterprise settings. In *Proceedings of the VLDB Endowment*, volume 17.

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., and Zhou, J. (2024). Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Lei, W., Wang, W., Ma, Z., Gan, T., Lu, W., Kan, M.-Y., and Chua, T.-S. (2020). Re-examining the role of schema linking in text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954. Association for Computational Linguistics.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K. C., Huang, F., Cheng, R., and Li, Y. (2024). Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Pourreza, M. and Rafiei, D. (2024). DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*.

Pydantic AI Team (2024). Pydantic AI: Agent framework for model serving. `https://ai.pydantic.dev/`.

Rajkumar, N., Li, R., and Bahdanau, D. (2022). Evaluating the text-to-SQL capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3911–3921. Association for Computational Linguistics.