Agentic Benchmark Generation and Factorial Evaluation of Text-to-SQL Architectures on an Industrial Database

Thiago Pedroso de Jesus[1,2], **Victor Arantes**[1,2], **Fernanda Färber**[1,2], **Artur Matos**[1,2]
Vinicius Aguiar[1,2], **Sávio Teles**[1,2], **Anderson da Silva Soares**[1,2]

[1]Institute of Informatics – Federal University of Goiás (UFG)
Goiânia – GO – Brazil

[2]Center of Excellence in Artificial Intelligence (CEIA) – Brazil

{thiagopedroso, victor_arantes, fernandabufon,
artur.matos, vinicius_aguiar2}@discente.ufg.br
savioteles@ufg.br, anderson@inf.ufg.br

Abstract. *Deploying Text-to-SQL in specialized industrial domains demands an evaluation benchmark grounded in real operational needs and an understanding of which architectural choices truly improve accuracy. Human-annotated benchmarks are prohibitively expensive, and current synthetic generators read only the schema, never the data, missing the quirks that make industrial questions hard. We present two contributions. First, an agentic benchmark generation pipeline in which a ReAct agent with five database-exploration tools formulates each question/SQL pair from observed data, enforcing quality through execution validation, vocabulary gates, and dual-layer deduplication; it expands 37 expert-validated seeds into a 169-pair Portuguese benchmark without paid annotation. Second, a controlled factorial study that fixes the model, benchmark, and metadata to isolate how schema presentation and tool access affect six architectures. We find that full-schema access and tool use add accuracy independently, that semantic schema linking helps simple but starves medium-complexity queries, and that self-correction and multi-agent decomposition do not pay off; above all, 57% of questions fail across every architecture, showing that the binding constraint is domain knowledge, not architectural sophistication.*

Keywords: Text-to-SQL; Agentic benchmark generation; Schema linking; Factorial evaluation; Industrial NLP; LLM agents.

1. Introduction

Natural language interfaces to relational databases support decision-making in asset-intensive industries. Engineers and managers at electric utilities need to query network topology, equipment status, and planning data without writing SQL, and LLM-based Text-to-SQL systems are being deployed for exactly this purpose [Wang et al. 2025, Xie et al. 2025]. Yet deploying such a system in a specialized industrial domain raises two intertwined challenges: how to build an evaluation benchmark that reflects real operational needs, and which architectural choices actually improve accuracy on that benchmark.

Constructing benchmarks through human annotation is expensive: the BIRD benchmark [Li et al. 2023] required roughly US$98,000 and 1,000 annotation hours for a general-

purpose English dataset, and reproducing this effort for each industrial domain is impractical. By contrast, the benchmark used here was produced without any paid annotation campaign: a single 61-minute expert session seeds an otherwise automated pipeline of LLM-driven exploration and augmentation (Section 3.4). Synthetic generation via LLMs has emerged as a scalable alternative [Li et al. 2025, Pan et al. 2026, Caferoğlu et al. 2025], yet all existing pipelines share a fundamental limitation: they use the database schema (table names, column types, and constraints) as their sole input, without ever querying the data. As a result, generated questions may reference non-existent values (*e.g.*, a value with trailing spaces), assume join semantics invisible in the schema, or produce operationally irrelevant aggregations. Relevance is not guaranteed by data grounding alone: in a validation session, two specialists rated 36% of a preliminary synthetic set as irrelevant for daily operations (Section 3.4).

On the architectural side, the literature offers many techniques (agentic tool use, semantic schema linking, self-correction, and multi-agent decomposition), yet controlled comparisons that isolate each component are scarce, as most evaluations change several variables at once (model, schema presentation, tool access), making it hard to determine why one system outperforms another. We address both challenges with two coupled contributions. **First**, an agentic benchmark generation pipeline that grounds each natural-language/SQL pair in observed data distributions through interactive database exploration, expanding expert-validated seed questions into a benchmark with controlled quality and complexity coverage. This addresses the schema-only blind spot of prior synthetic generators. **Second**, a controlled factorial evaluation that isolates the contribution of two architectural axes (schema presentation and tool access) by holding the LLM, benchmark, and metadata fixed across architectures, complemented by self-correction and multi-agent variants.

We instantiate the system on a real relational database from a Brazilian electric utility (14 tables, 230 columns, ~179k rows in the central table, queries in Portuguese), contrasting with the English general-domain focus of Spider [Yu et al. 2018] and BIRD [Li et al. 2023]. The evaluation, using both Execution Accuracy and FLEX [Kim et al. 2025] as an LLM-as-judge complement, yields four findings that bear on architecture choice in this regime (a small in-context schema, a single Portuguese industrial database, and a single evaluated model family): (i) performance decomposes into two additive factors (full schema: $+2.4$ pp; tool use: $+4.1$ pp); (ii) semantic schema linking improves simple queries but causes information starvation on medium-complexity ones (-11.6 pp); (iii) self-correction and multi-agent decomposition yield zero net gains at higher cost; and (iv) 57% of questions fail across all six architectures, pointing to domain knowledge as the binding constraint, not architectural sophistication.

2. Related Work

2.1. Text-to-SQL Benchmark Construction

Text-to-SQL benchmarks have traditionally relied on costly human annotation of general-purpose, multi-domain, English datasets. Spider [Yu et al. 2018] and BIRD [Li et al. 2023] are the canonical examples, built from thousands of annotation hours with execution-result validation. Because their schemas and questions are general-purpose and in English, they transfer poorly to specialized industrial domains without substantial re-annotation, which motivates cheaper synthetic alternatives.

To reduce this cost, LLM-based synthetic generation follows two directions. *SQL-first* methods generate SQL from the schema and then a question: OmniSQL [Li et al. 2025] scales this to 2.5M pairs, and EvolSQL [Pan et al. 2026] mutates SQL abstract syntax trees to reach 65.1% on BIRD with 1/18 of the data. *Question-first* methods [Cheng et al. 2025] instead start from questions and synthesize SQL afterwards, better preserving phrasing; SING-SQL [Caferoğlu et al. 2025] adds hierarchical schema partitioning and LLM-as-judge filtering with automatic repair.

Despite their differences, all these methods share a fundamental limitation: they operate on the schema alone, without executing queries or examining actual data. This leads to questions based on non-existent values, incorrect JOIN assumptions, and domain-irrelevant aggregations. Our benchmark agent closes this gap by actively exploring the database before formulating each question.

2.2. Agentic Approaches

Agentic reasoning has been applied to the Text-to-SQL *execution* problem, where the agent explores the database to improve a single answer: SDE-SQL [Xie et al. 2025] issues exploratory probes to reveal value distributions, RAISE [Granado et al. 2025] validates data hypotheses through execution, APEX-SQL [Cao et al. 2026] couples exploration with iterative refinement, and MAC-SQL [Wang et al. 2025] decomposes the task across specialized agents.

The agentic paradigm has also been applied to synthetic data generation in other domains, *e.g.*, AgentInstruct [Mitra et al. 2024] produces instruction pairs with tool-equipped agents, and related agents synthesize tasks grounded in explored environment states, directly analogous to exploring database contents.

None of these works apply agentic exploration to Text-to-SQL benchmark generation, and controlled comparisons isolating individual components (*e.g.*, tool access vs. schema presentation) are rare.

2.3. Schema Linking and LLM-Generated Metadata

Schema linking, the task of identifying relevant tables and columns for a given question, is a recurring lever for Text-to-SQL accuracy. CHESS [Talaei et al. 2024] formalizes four linking mechanisms (metadata search, column filtering, table and column selection); RSL-SQL [Cao et al. 2024] proposes bidirectional linking that achieves 67.2% on BIRD; AutoLink [Wang et al. 2026] scales to schemas with 3,000+ columns while maintaining 97.4% recall. On the metadata side, Shkapenyuk et al. [Shkapenyuk et al. 2025] show that automatically generated profiling metadata outperforms human annotations on BIRD, and Cocoon [Huang and Wu 2024] infers column semantics from data distributions; in-context methods such as DIN-SQL [Pourreza and Rafiei 2023] and DAIL-SQL [Gao et al. 2024] add decomposed prompting and schema masking. However, schema linking is typically evaluated on large schemas that exceed the context window; whether it helps or hurts on *small* schemas that fit entirely in context has received little attention, which our factorial comparison addresses.

For Portuguese, Jose et al. [José and Cozman 2021] adapted mRAT-SQL+GAP for Brazilian Portuguese Text-to-SQL, an early precedent for the task. Our work extends this direction to industrial domains with agentic architectures.

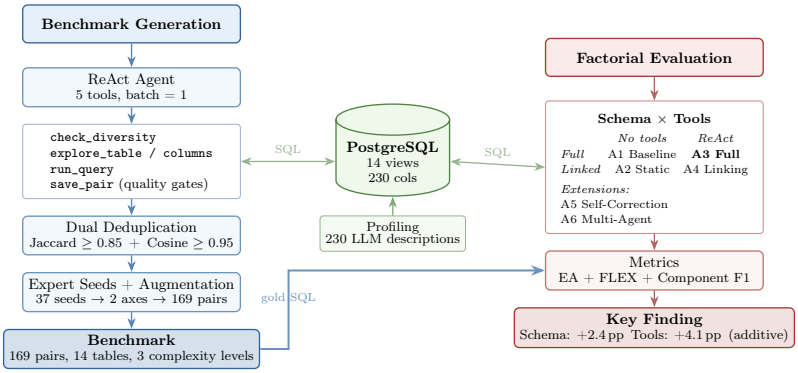


Figure 1. System overview. Left: the benchmark agent explores the database before generating each NL/SQL pair, with a two-axis augmentation pipeline expanding 37 seeds into 169 pairs. Right: six architectures varying schema presentation and tool access are evaluated on the benchmark.

2.4. Positioning

Rather than propose a new benchmark generator or a new agent architecture, this work composes existing components (ReAct exploration, semantic schema linking, and profile-based metadata) into a pipeline that admits controlled comparison on a regime rarely studied jointly: a small industrial relational schema in Portuguese, with questions grounded in observed data semantics. The combination is what distinguishes this work: it explores the database during construction (unlike schema-only generators such as OmniSQL [Li et al. 2025] and EvolSQL [Pan et al. 2026]), it yields a reusable Text-to-SQL benchmark (unlike execution-time agents such as SDE-SQL [Xie et al. 2025] and MAC-SQL [Wang et al. 2025]), it supports a factorial architectural evaluation, and it targets a real industrial domain in Portuguese rather than general-purpose English data [Yu et al. 2018, Li et al. 2023].

3. Proposed System

To address the challenges of benchmark construction and architecture selection for industrial Text-to-SQL, we propose a system with three components: (i) a profiling and metadata extraction pipeline, (ii) a benchmark generation and augmentation pipeline, and (iii) a family of Text-to-SQL architectures that vary in schema presentation and tool access. All components share the same database and metadata infrastructure.

Figure 1 illustrates the overall system. The left side shows the offline benchmark construction: a ReAct agent explores the database before formulating each NL/SQL pair, and an augmentation pipeline expands expert-validated seeds into the final benchmark. The right side shows the online evaluation: six Text-to-SQL architectures varying in schema presentation and tool access run on the same benchmark.

3.1. Dataset and Domain

The target database belongs to the distribution-network directorate of a Brazilian electric utility. It consists of 14 relational tables covering network topology, substations, equipment, transformers, poles, planned works, primary customers, and expansion planning. Table sizes range from a few hundred rows to roughly 179k in the central medium-voltage network table, and the schema spans 230 columns in total.

The database exhibits domain-specific peculiarities that are invisible in the schema definition but discoverable through data exploration, such as trailing spaces in join keys, encoding corruption in text fields, and implicit foreign-key relationships without explicit constraints. These quirks motivate the agentic approach: only by querying the actual data can a system discover and handle them correctly.

3.2. Profiling and Metadata Extraction

Inspired by Shkapenyuk et al. [Shkapenyuk et al. 2025] and Cocoon [Huang and Wu 2024], we generate column metadata once offline in two phases. Phase 1 computes per-column statistics (row, null, and distinct counts; top values with frequencies; min/max/avg; mean string length) and infers heuristic join hints from same-named columns whose value overlap exceeds 50%. Phase 2 has an LLM summarize each column's statistics and table context into a one-to-two sentence description, yielding 230 descriptions that also surface undocumented domain knowledge such as encoding anomalies and implicit status-code semantics. Table and column embeddings (`text-multilingual-embedding-002`, 768 dims) are indexed for cosine-similarity retrieval.

3.3. Benchmark Generation Agent

The benchmark agent follows the ReAct pattern [Yao et al. 2023] and handles exactly one question per instance (batch size = 1), since preliminary experiments showed systematic degradation when generating multiple questions per session: the fifth question averaged 1–2 tool calls versus 3–4 for the first. The system prompt instructs the agent to adopt the persona of a distribution-network director asking operational questions, avoid database jargon, generate self-contained questions, and validate plausibility before saving.

The agent has access to five tools: `check_diversity` (table coverage, SQL operations used, and recent questions), `explore_table` (DDL, row count, column list), `explore_columns` (per-column profiling: top values, null rate, distinct count, min/max/avg), `run_query` (executes SQL with a 30 s timeout and auto-LIMIT), and `save_pair` (applies the quality gates below). The agent typically checks diversity, inspects tables and columns, probes with exploratory queries, formulates a question, validates it by execution, and saves.

The `save_pair` tool enforces four quality gates before accepting a pair: (1) execution validation, requiring that `run_query` was called with the final SQL; (2) length limit (≤ 300 characters); (3) vocabulary blacklist, rejecting column names, internal codes, and verbatim sample values; and (4) structural deduplication via SQL fingerprint Jaccard similarity (≥ 0.85 triggers rejection). A secondary offline pass deduplicates questions whose embedding cosine similarity is ≥ 0.95 (union-find); this conservative threshold, higher than the 0.7–0.9 used by SemDeDup [Abbas et al. 2023] for web-scale data, is justified by the restricted domain vocabulary (at 0.90, 59% of flagged pairs were false positives). Each question is also classified programmatically following Spider [Yu et al. 2018] into simple ($\leq$4), medium (5–7), or complex ($\geq$8) based on SQL component scores.

3.4. Expert Seeds and Data Augmentation

Automatic generation produces fluent synthetic pairs but cannot guarantee operational relevance: the 36% irrelevance rate motivating this work was measured on a preliminary

synthetic set assembled before the pipeline of Section 3.3. We therefore complement automatic generation with a two-stage human-grounded process: a validation session with domain experts that yields a small set of authoritative seed questions, followed by an augmentation pipeline that scales those seeds into the final benchmark.

Expert validation. A 61-minute session with two distribution-network specialists (planning engineer and field maintenance engineer) assessed 25 of those questions on a 1–5 scale. Evaluators also reformulated rejected questions and described real query patterns. Cross-referencing annotations with the session transcript recovered 12 additional queries mentioned verbally, a 29% increase over annotation-only extraction. Of 42 candidates (9 validated originals, 10 reformulations, 11 corrective suggestions, 12 from transcript), 37 were included after SQL validation.

Two-axis augmentation. The augmentation pipeline expands the 37 seeds into 169 pairs via two parallel axes scheduled by a deterministic planner. **Axis 1 (SQL-first, 43%)** applies one of seven structural operations to the seed SQL (*e.g.*, add join, add table, change filter/metric/granularity, increase or decrease complexity), validates it by execution, and writes a matching question (structural Jaccard gate ≥ 0.85). **Axis 2 (question variation, 57%)** holds the SQL fixed and rewrites the question in one of eight styles (formal, informal, descriptive, concise, vague, reasoned, imperative, paraphrase), gated by ROUGE-L < 0.7 against existing variations. The result is 169 pairs (37 seeds+132 augmented) with 100% SQL validation and all 14 tables covered. No augmented pair received individual expert review: Axis 2 preserves the validated information need by construction, whereas Axis 1 yields new queries validated only by execution.

3.5. Text-to-SQL Architectures Under Comparison

To isolate the effect of architectural choices, we define six architectures that systematically vary schema presentation (full schema vs. semantic linking) and tool access (single-call vs. ReAct agent), plus two additional variants (self-correction and multi-agent). All six use the same LLM (Gemini 3 Flash Preview, temperature 0.0), benchmark, and metadata package (LLM descriptions, distinct/null counts, top-3 values, JOIN hints), so any A3 versus A4 gap reflects schema presentation, not metadata density.

A1 (Baseline) injects the full 14-table schema and generates SQL in a single call. **A2 (Static Linking)** retrieves the top-5 tables and top-15 columns by embedding similarity before a single call. **A3 (ReAct Full)** is a ReAct agent [Yao et al. 2023] with full-schema access and three tools (`get_tables`, `get_schema`, `run_query`), averaging 7.5 tool calls per question. **A4 (ReAct Linking)** replaces the full-schema tools with semantic linking: `table_retriever` ranks tables by cosine similarity and `get_columns_info` returns the top-15 columns plus mandatory JOIN and token-matched columns, enriched with descriptions, statistics, and top values. **A5 (Self-Correction)** extends A4 with a Reflexion-style reflection loop of up to three iterations [Shinn et al. 2023]. **A6 (Multi-Agent)**, inspired by MAC-SQL [Wang et al. 2025], splits a Planner (semantic linking, structured plan) from an Executor (`run_query`). The four architectures A1–A4 form a clean 2×2 factorial design crossing schema presentation with tool access; A5 and A6 probe whether self-correction and task decomposition add value beyond the ReAct baseline.

4. Experiments and Results

4.1. Experimental Setup

Setup. All six architectures use Gemini 3 Flash Preview (temperature 0.0) against the same database and metadata package (Appendix). The benchmark contains 169 question/SQL pairs from 37 expert-validated seeds (Section 3.4), of which 94 are unique SQL queries (75 are paraphrases of the same gold); complexity is 104 simple (62%), 43 medium (25%), 22 complex (13%).

Metrics. We report Execution Accuracy (EA) [Yu et al. 2018], comparing result sets with order-insensitive, numerically tolerant ($\pm$1e-4), case-insensitive matching; FLEX [Kim et al. 2025], an LLM-as-judge metric (judge: GPT-5.2, cross-family) that catches EA false positives and false negatives; and component-level F1 (tables, columns, operators, aggregations).

Statistical note. With $n = 169$ and 75 paraphrase pairs, a 4.1 pp difference is $\sim$7 questions. Paired McNemar tests find no architecture pair differing significantly (two-sided exact $p = 0.11$–0.48; *e.g.*, A3 vs. A4: 11 vs. 7 discordant, $p = 0.48$), so we read the factorial decomposition as a structured pattern and a cost–accuracy trade-off rather than a separated ranking. Re-evaluating on the 94 unique-SQL queries (collapsing each paraphrase group) for the three architectures with retained logs shifts EA by under 1.5 pp (A2 $29.6\rightarrow30.1$, A4 $33.7\rightarrow33.6$, A6 $29.0\rightarrow29.1$) and preserves the ordering, so paraphrase redundancy does not materially inflate the metrics (60 of 94 unique queries are singletons).

4.2. Benchmark Generation Results

Enabling all quality gates with batch size 1, the agent processed 167 attempts to yield 100 validated questions: a 40% online rejection rate (12 execution failures, 30 oversized, 11 vocabulary violations, and 14 structural duplicates concentrated in the second half of the run, confirming saturation) and zero post-hoc violations, in contrast to a no-gate configuration that admitted semantic duplicates, oversized, and vocabulary-violating questions. Augmentation then produced 132 pairs from the 37 seeds (57 via Axis 1, 75 via Axis 2), covering all augmentation types with 100% SQL validity and mean intra-seed ROUGE-L of 0.307 (strong lexical diversity).

4.3. Main Results: Architecture Comparison

Table 1 presents the central results. A3 (ReAct, full schema) tops both metrics (36.1% EA, 45.0% FLEX), while the one-call baseline A1 reaches 32.0% EA at 29% of the cost. The Pred% column exposes A2's brittleness: at 90.5%, roughly one output in ten fails to run, versus $\geq$98.8% for every other arm. The ranking is unpacked below.

Factorial decomposition. The four architectures A1–A4 form a 2×2 factorial design. On EA the 2×2 decomposition is additive: tool use contributes $+4.1$ pp regardless of schema mode (Full $32.0\rightarrow36.1$, Linked $29.6\rightarrow33.7$), and full-schema access contributes $+2.4$ pp regardless of tool access. On FLEX an interaction emerges: tools help more under semantic linking ($+10.1$ pp: $33.1\rightarrow43.2$) than under full schema ($+5.9$ pp: $39.1\rightarrow45.0$), as

Table 1. Main results across six architectures on 169 questions. Cost in BRL for the full evaluation run; kTok/Q is mean thousands of tokens per question (input + output). Best values in bold.

Arch.	EA	FLEX	F1	Pred%	Cost	kTok/Q	Calls/Q
A1 Baseline	32.0	39.1	.714	98.8	R$ 8	31	1.0
A2 Static Link	29.6	33.1	.684	90.5	R$ 2	10	1.0
A3 ReAct Full	**36.1**	**45.0**	**.733**	99.4	R$ 28	107	8.5
A4 ReAct Link	33.7	43.2	.721	100	R$ 50	193	10.6
A5 Self-Corr.	36.1	43.2	.723	99.4	R$ 32	125	9.3
A6 Multi-Agent	29.0	34.3	.725	98.8	R$ 21	81	8.9

Table 2. Execution Accuracy (%) by complexity level. Best per column in bold.

Architecture	Simple *(n=104)*	Medium *(n=43)*	Complex *(n=22)*
A1 Baseline	41.3	18.6	13.6
A2 Static Linking	39.4	11.6	18.2
A3 ReAct Full	45.2	**18.6**	**27.3**
A4 ReAct Linking	**47.1**	7.0	22.7
A5 Self-Correction	**49.0**	14.0	18.2
A6 Multi-Agent	36.5	14.0	22.7

interactive exploration partly compensates for filtered-out information; conversely, without tools, full schema dominates ($+6.0$ pp) since filtered information cannot be recovered. Tools contribute mainly through iterative validation (A3 issues 4.3 `run_query` calls per question on average), which is especially valuable when schema information is incomplete.

EA vs. FLEX gap. FLEX reveals 3.5–9.5 pp more correct SQL than EA, with a larger delta for agentic architectures ($\sim$9 pp) than single-call ones: agents produce SQL that differs from the gold only in cosmetic details (aliases, ORDER BY direction). The consistent $\sim$10% false-positive baseline points to intrinsic gold-SQL ambiguities rather than architectural artifacts.

4.4. Schema Linking

Breaking EA down by query complexity (Table 2) exposes a trade-off that depends on query difficulty.

On **simple queries**, A4 (47.1%) and A5 (49.0%) outperform A3 (45.2%): focusing the agent on the most relevant columns reduces distraction when the query involves a single table. On **medium queries**, A4 collapses to 7.0%, below the zero-tool A1 (18.6%) and even A2 (11.6%). This is information starvation: medium queries often require columns that the semantic retriever does not rank in the top-K, preventing the agent from generating correct JOINs or filters. On **complex queries**, full-schema access matters most (A3: 27.3%), because complex JOINs and multi-table aggregations require columns the retriever may miss.

A concrete instance illustrates the starvation mechanism. For a medium query counting in- and out-of-service equipment per municipality, A3 (full schema) correctly filters on `estado` (operational state in the network), whereas A4 (linked) filters on `posope` (physical operating position): two columns sharing the codes `'F'/'A'` but with different semantics. With the full schema, the agent sees both columns' descriptions and top values side by side and disambiguates; under semantic linking the retriever surfaces `posope` as most relevant to "operation" and hides `estado`, producing a plausible but wrong query.

For the four linking architectures (A2, A4, A5, A6), Table Recall ≈0.74 and Column Recall ≈0.76: the retriever finds the right material in ∼75% of cases, so the bottleneck is SQL generation, not retrieval.

This complements positive schema-linking results on large schemas [Li et al. 2023, Gao et al. 2024]: when the schema fits in context, linking creates a problem (information loss) rather than solving one (overflow).

4.5. Self-Correction and Multi-Agent Decomposition

Self-correction (A5). A5 recovers some of A4's medium-query losses ($+7.0$ pp) but degrades complex queries (-4.5 pp) at 17% higher cost, while topping simple-query accuracy (49.0%). At the question level it corrects 11 of A4's errors but introduces 7 new ones, an over-correction pattern concentrated on complex queries.

Multi-agent (A6). A6 underperforms A3 by 7.1 pp EA and even falls below the baseline A1 by 3.0 pp. The Planner identifies correct operations (best Operator F1: 0.713), but the Executor, receiving a structured plan without full exploration context, issues 58% fewer validation queries (302 vs. 727 for A3) and loses contextual nuances such as exact filter values and JOIN patterns.

4.6. The Domain-Knowledge Bottleneck

Across the benchmark, 97 questions (57%) fail in all six architectures, concentrating on queries that require undocumented business rules, *e.g.*, urban/rural classification via `cli_por_km` > 10 in a separate table reachable only through an implicit join, or the `estado`/`posope` code distinction discussed above. No architectural sophistication compensates for absent domain knowledge, suggesting that schema augmentation with business rules would yield higher returns than pipeline complexity.

5. Conclusion

We presented an agentic benchmark generation pipeline that explores a database before formulating each question, and used the resulting 169-pair benchmark for a controlled factorial comparison of six Text-to-SQL architectures. The evaluation yielded four findings that bear on architecture choice in this regime: performance decomposes into two additive factors on EA (full schema: $+2.4$ pp; tools: $+4.1$ pp); semantic schema linking causes information starvation on medium-complexity queries (-11.6 pp); self-correction and multi-agent decomposition yield no net gains; and 57% of questions fail universally, pointing to domain knowledge as the binding constraint.

Limitations. These results characterize a specific regime and should not be read as universal rules for Text-to-SQL architecture design. First, the evaluation covers a single industrial database (14 tables, Portuguese) and a single evaluated LLM family (Gemini 3 Flash Preview), held fixed by design to isolate architectural effects; the conclusions may not transfer to other model families or larger schemas where context overflow makes schema linking essential. Second, the differences between architectures are small relative to the benchmark size: paired McNemar tests find no architecture pair separating significantly ($p > 0.10$), so we report the factorial decomposition as a structured pattern and a cost–accuracy trade-off rather than a statistically separated ranking. Third, the benchmark contains paraphrase redundancy (75 of 169 questions share gold SQL); re-evaluating on the 94 unique-SQL queries shifts execution accuracy by less than 1.5 pp and preserves the ordering (Section 4.1), but the redundancy still limits the effective sample. Fourth, the benchmark was generated by our own pipeline and validated by two domain experts on a subset of questions; independent external validation, and comparison against an external Text-to-SQL baseline adapted to this domain, would strengthen confidence in its representativeness.

Future work. Promising directions include: (i) enriching metadata with explicit business rules (the identified bottleneck) and measuring the accuracy gain; (ii) extending the benchmark generation pipeline to other industrial domains and languages; (iii) evaluating on larger schemas where the schema-linking trade-off may reverse; (iv) investigating hybrid approaches that use full schema for complex queries and linked schema for simple ones; and (v) replicating the factorial study across additional model families and against an external Text-to-SQL baseline to test whether the present findings hold beyond this regime.

Acknowledgments

This work has been funded by P&D CEMIG/ANEEL PD04950-D0677/2023 supported by the Center of Excellence in Artificial Intelligence (CEIA), with financial resources from the MCTI grant number 057/2023, signed with EMBRAPII. This work was also supported by the National Institute of Science and Technology (INCT) in Responsible Artificial Intelligence for Computational Linguistics and Information Treatment and Dissemination (TILD-IAR) grant number 408490/2024-1.

Disclosure of Interests

The authors declare no competing interests.

Appendix: Reproducibility Details

The benchmark agent uses Gemini 3.1 Pro Preview; the six evaluation architectures use Gemini 3 Flash Preview at temperature 0.0; the FLEX judge is GPT-5.2 (cross-family). Embeddings use `text-multilingual-embedding-002` (768 dims, Vertex AI). The linking architectures retrieve the top-5 tables and top-15 columns by cosine similarity, always adding mandatory JOIN columns and respecting a per-column visibility flag; A5 runs up to three reflection iterations. Every architecture receives the same metadata

package (data types, LLM descriptions, distinct/null counts, top-3 values, JOIN hints) and a shared base prompt: emit only `SELECT`/`WITH` over `pub.*` views, return the minimal SQL answering exactly the question, avoid gratuitous `TRIM`/`CAST`/`ROUND`, and prefer the simplest reading of ambiguous questions, with architecture-specific blocks for schema presentation and tool use. Experiments use Agno 2.5.15, Python 3.12, and `psycopg2` 2.9.11 against PostgreSQL (Cloud SQL) with pgvector; costs are in BRL at USD/BRL$=5.0042$ and the benchmark is fixed by content hash.

References

[Abbas et al. 2023] Abbas, A., Tirumala, K., Simig, D., Ganguli, S., and Morcos, A. S. (2023). SemDeDup: Data-efficient learning at web-scale through semantic deduplication. *arXiv preprint arXiv:2303.09540*.

[Caferoğlu et al. 2025] Caferoğlu, H. A., Çelik, M. S., and Ulusoy, O. (2025). SING-SQL: A synthetic data generation framework for in-domain text-to-SQL translation. *arXiv preprint arXiv:2509.25672*.

[Cao et al. 2026] Cao, B., Liao, W., Sun, Y., Fang, D., Li, H., and Lam, W. (2026). APEX-SQL: Talking to the data via agentic exploration for text-to-SQL. *arXiv preprint arXiv:2602.16720*.

[Cao et al. 2024] Cao, Z., Zheng, Y., Fan, Z., Zhang, X., Chen, W., and Bai, X. (2024). RSL-SQL: Robust schema linking in text-to-SQL generation. *arXiv preprint arXiv:2411.00073*.

[Cheng et al. 2025] Cheng, S., Cheng, Q., Jin, L., Yi, L., and Zhang, G. (2025). SQLord: A robust enterprise text-to-SQL solution via reverse data generation and workflow decomposition. In *Companion Proceedings of WWW 2025*, pages 919–923. Association for Computing Machinery.

[Gao et al. 2024] 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.

[Granado et al. 2025] Granado, F., Lotufo, R., and Pereira, J. (2025). RAISE: Reasoning agent for interactive SQL exploration. *arXiv preprint arXiv:2506.01273*.

[Huang and Wu 2024] Huang, Z. and Wu, E. (2024). Cocoon: Semantic table profiling using large language models. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics (HILDA)*, pages 1–7. Association for Computing Machinery.

[José and Cozman 2021] José, M. A. and Cozman, F. G. (2021). mRAT-SQL+GAP: A portuguese text-to-SQL transformer. In *Proceedings of the Brazilian Conference on Intelligent Systems (BRACIS)*, volume 13074 of *LNAI*, pages 511–525. Springer.

[Kim et al. 2025] Kim, H., Jeon, T., Choi, S., Choi, S., and Cho, H. (2025). FLEX: Expert-level false-less execution metric for text-to-SQL benchmark. In *Proceedings of NAACL 2025*, pages 4448–4475. Association for Computational Linguistics.

[Li et al. 2025] Li, H., Wu, S., Zhang, X., Huang, X., Zhang, J., Jiang, F., Wang, S., Zhang, T., Chen, J., Shi, R., Chen, H., and Li, C. (2025). OmniSQL: Synthesizing high-quality text-to-SQL data at scale. *arXiv preprint arXiv:2503.02240*.

[Li et al. 2023] Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Chenhao, M., Li, G., Chang, K., Huang, F., Cheng, R., and Li, Y. (2023). 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)*, volume 36, pages 42330–42357.

[Mitra et al. 2024] Mitra, A., Del Corro, L., Zheng, G., Mahajan, S., Rouhana, D., Codas, A., Lu, Y., Chen, W.-g., Vrousgos, O., Rosset, C., Silva, F., Khanpour, H., Lara, Y., and Awadallah, A. (2024). AgentInstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*.

[Pan et al. 2026] Pan, X., Tao, C., Bai, J., Gao, J., Tao, Z., Zhou, X., Cheung, G., and Ma, S. (2026). EvolSQL: Structure-aware evolution for scalable text-to-SQL data synthesis. *arXiv preprint arXiv:2601.04875*.

[Pourreza and Rafiei 2023] Pourreza, M. and Rafiei, D. (2023). DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 36339–36348.

[Shinn et al. 2023] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 8634–8652.

[Shkapenyuk et al. 2025] Shkapenyuk, V., Srivastava, D., Johnson, T., and Ghane, P. (2025). Automatic metadata extraction for text-to-SQL. *arXiv preprint arXiv:2505.19988*.

[Talaei et al. 2024] Talaei, S., Pourreza, M., Chang, Y.-C., Mirhoseini, A., and Saberi, A. (2024). CHESS: Contextual harnessing for efficient SQL synthesis. *arXiv preprint arXiv:2405.16755*.

[Wang et al. 2025] Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q.-W., Yin, D., Sun, X., and Li, Z. (2025). MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of COLING 2025*, pages 540–557. Association for Computational Linguistics.

[Wang et al. 2026] Wang, Z., Zheng, Y., Cao, Z., Zhang, X., Wei, Z., Fu, P., Luo, Z., Chen, W., and Bai, X. (2026). AutoLink: Autonomous schema exploration and expansion for scalable schema linking in text-to-SQL at scale. In *Proceedings of AAAI 2026*. AAAI Press.

[Xie et al. 2025] Xie, W., Dai, Y., and Jiang, W. (2025). SDE-SQL: Enhancing text-to-SQL generation in large language models via self-driven exploration with SQL probes. *arXiv preprint arXiv:2506.07245*.

[Yao et al. 2023] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

[Yu et al. 2018] 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.