Agentic Text-to-SQL in Enterprise Environments: Balancing Planning Paradigms and Pragmatic Self-Repair

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

[1]Grupo Visagio

{renato.miyaji,vitor.bedin,joao.macedo, renato.moulin}@visagio.com

Abstract. *The rapid adoption of Large Language Models (LLMs) has revolutionized Text-to-SQL systems, yet translating queries in complex enterprise environments remains challenging due to schema hallucinations and domain-specific jargon. We propose a multi-agent architecture that replaces internal generative self-correction with a database-driven self-repair loop and a two-tier Retrieval-Augmented Generation (RAG) system. We evaluate multiple planning and tool-use methods using a real-world corporate dataset. Our findings reveal that dual-system reasoning architectures provide an optimal balance of syntactic precision and user intent alignment. Furthermore, heavy iterative self-refinement introduces latency without proportional execution gains. This work establishes guidelines for designing reliable Text-to-SQL agents by decoupling cognitive planning from deterministic technical execution.*

1. Introduction

The rapid adoption of Large Language Models (LLMs) has revolutionized natural language interfaces for relational databases, widely known as Text-to-SQL systems [Chen et al. 2023]. In complex enterprise environments, translating user queries into precise SQL statements remains a challenge [Katsogiannis-Meimarakis and Koutrika 2023]. These systems frequently struggle with domain-specific jargon, complex table structures, and a high susceptibility to schema and value hallucinations [Talaei et al. 2024]. Ensuring correctness and operational robustness requires more than just zero-shot generation; it demands an alignment between the user's intent and the underlying database architecture [Shi et al. 2025]. In this context, Text-to-SQL transitions from being an isolated generation to a specialized tool within a broader multi-agent workflow. This architectural shift requires agents to not only comprehend natural language but also to effectively plan tool invocations, manage arguments, and navigate the syntactic constraints of structured query languages [Pourreza et al. 2025].

A critical component of Agentic Text-to-SQL is the underlying planning mechanism, often categorized into System 1 (fast, intuitive) and System 2 (deliberative, analytical) reasoning [Kahneman 2011]. While baseline models like ReAct interleave reasoning and acting [Yao et al. 2023], they often fail to grasp the holistic schema required for complex queries. Conversely, decoupled planning architectures like ReWOO separate strategic planning from execution [Xu et al. 2023], and dual-system approaches like SwiftSage dynamically route queries to either a fast-response node or a deliberate planning node [Lin et al. 2023]. Understanding how these distinct cognitive blueprints impact the accuracy and latency of SQL generation is vital for deploying reliable industrial applications.

Furthermore, reliability in LLM-driven systems is frequently pursued through iterative refinement techniques, such as Reflexion [Shinn et al. 2023] and Self-Refine [Madaan et al. 2023], which utilize internal critique loops to improve outputs. However, applying these introspective methods to Text-to-SQL introduces a significant challenge: cognitive conflict and latency overheads. When an agent attempts to self-correct based on its own internal reasoning, it often conflicts with the execution-based feedback provided by the database management system itself. This redundancy can propagate errors rather than resolve them, highlighting the need for pragmatic self-repair loops [Liu et al. 2025].

In this paper, we propose and evaluate a robust Text-to-SQL architecture designed for agentic environments. Our system implements a two-tier Retrieval-Augmented Generation (RAG) approach: a structural layer that injects dynamic few-shot SQL examples [Brown et al. 2020], and a semantic layer that utilizes extraction and similarity matching for precise value grounding [Zhang et al. 2026]. Instead of relying on the LLM's internal critique for error correction, the architecture employs a self-repair loop that feeds actual database execution errors back into the prompt, fortified by strict pre-execution guardrails.

Leveraging this architecture, we conduct an empirical analysis to evaluate the efficacy of various planning and tool-use paradigms—including ReAct, ReWOO, SwiftSage, Multiplan (Self-Consistency-based methods) [Wang et al. 2023], Reflexion, Self-Refine, Dynamic Few-Shot Learning [Brown et al. 2020], and ToolVerifier [Mekala et al. 2024]—in generating correct and executable SQL queries. By assessing metrics such as SQL Correctness [Shi et al. 2025], Relaxed Execution Accuracy [Pinna et al. 2025], Human Evaluation, and Latency, our objective is to isolate the impact of cognitive planning quality from technical execution. Our findings reveal that while dual-system reasoning provides an optimal balance of low latency and high syntactic precision, heavy iterative self-correction imposes prohibitive computational costs without yielding proportional execution gains.

The main contributions of this work are fourfold. First, we introduce a production-ready Text-to-SQL architecture featuring two-tier RAG and database-driven self-repair. Second, we provide a comparative evaluation of System 1 and System 2 planning paradigms, demonstrating the superiority of dual-reasoning models for precise SQL generation. Third, we empirically identify and map the cognitive conflicts and latency overheads associated with LLM-guided iterative refinement when applied to structured query tasks. Finally, we establish evidence-based guidelines for designing reliable, low-latency Text-to-SQL agents in enterprise environments, emphasizing the necessity of decoupling cognitive planning from deterministic technical execution, validated through a case study in a consulting company.

2. Related Works

2.1. Agentic Planning and Schema Assimilation

Traditional agentic paradigms, such as ReAct [Yao et al. 2023], often struggle when applied directly to complex database environments. Recent industry insights highlight that simple reactive loops frequently fail because they attempt to execute queries before fully validating the schema or determining the correct operational order [Eckmann et al. 2025]. To mitigate these errors, recent research has shifted towards multi-agent workflows. A primary strategy to reduce schema and value hallucinations is the explicit separation of

cognitive planning from SQL generation. For instance, SQL-of-Thought demonstrates that decoupling the generation of an execution plan from the actual SQL writing reduces hallucinations and imposes logical structure [Chaturvedi et al. 2024]. This is aligned with the investigation into whether decoupled planning and dual-reasoning architectures outperform baseline reactive models by isolating schema complexity made in this paper. Furthermore, assimilating complex corporate schemas requires targeted approaches. Recent works have proposed agentic exploratory models, such as APEX-SQL, which actively validate database schemas through hypothesis-checking loops rather than relying on static definitions [Cao et al. 2026]. Similarly, efforts focused on Small Language Models emphasize that rigorous schema-linking is essential for reducing table-level hallucinations [Cao et al. 2026].

2.2. RAG Constraints in Corporate Environments

Deploying NL2SQL in enterprise settings introduces requirements for context retrieval and data governance. Traditional RAG methodologies often fail to capture the complex relational nature of corporate databases. Research by Baptista & Garcia (2025) [Baptista and Garcia 2025] highlights these failures, specifically in complex domains, suggesting that purely semantic RAG must be augmented or replaced by relation-aware approaches. Additionally, corporate environments demand rigorous orchestration to prevent unauthorized data access. Martinelli & Ponti (2025) [Martinelli and Ponti 2025] argue that secure Retrieval-Augmented Generation requires complex role-based access control (RBAC) to prevent LLM data leaks. These findings justify our architectural choices: instead of naive document retrieval, our system employs a two-layer RAG approach to maintain deterministic control over the prompt context.

2.3. Iterative Refinement and Cognitive Conflicts

A critical challenge in modern agentic Text-to-SQL is error recovery [Chen et al. 2023]. While purely generative self-correction frameworks have gained popularity, their efficacy in rigid environments like SQL execution is debated. Recent studies analyzing the degradation of self-refinement models provide an explanation for this phenomenon. The ErrorLLM framework demonstrates that when LLMs attempt to self-correct SQL over multiple iterations without deterministic external guidance, they often generate false-positive error detections [Hong et al. 2026]. This leads to widespread query corruption, where the model generates false-positive error detections and breaks previously valid logic. This directly correlates with the cognitive conflicts explored in this work. When agent-guided iterative correction operates simultaneously with native Database Management System (DBMS) error feedback, the LLM's verbal reflection often attempts to reinterpret the user's intent rather than simply fixing minor syntax errors [Kamoi et al. 2024].

3. Methods

3.1. Proposed Method

In this section, we present an agentic Text-to-SQL architecture designed to navigate complex enterprise databases, presented in Figure 1. Unlike traditional reactive loops that often struggle with premature execution, our system employed an implicit planning framework orchestrated by an event-driven agent model. Acting as a specialized data analyst, the base agent leveraged dynamic context and followed a structured heuristic (interpret,

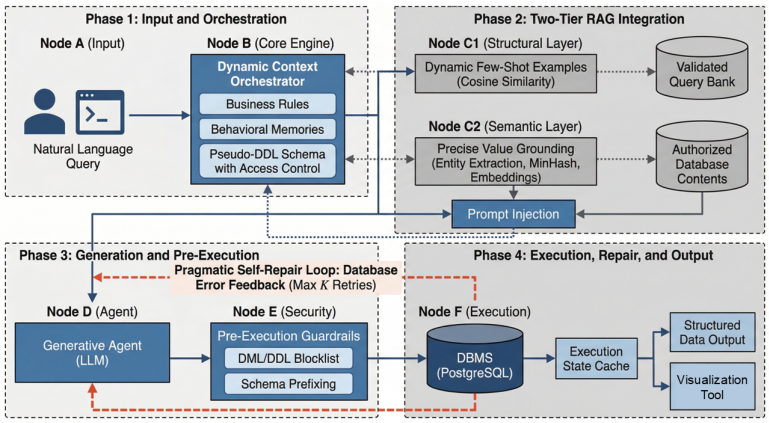


Figura 1. Proposed agentic Text-to-SQL architecture for complex enterprise databases

decompose, retrieve, analyze, and present) to process natural language queries without relying on a rigid planning loop. Based on this baseline architecture, we incorporated the various explicit planning paradigms and tool-use techniques to conduct our experiments.

A core component of our architecture is the Dynamic Context Orchestrator, which aggregates and injects layers of information into the prompt prior to LLM inference. This orchestrated context goes beyond basic prompting by embedding project-specific business rules and behavioral memories retrieved via RAG to guide the model using historical correction patterns. Crucially, it injects the permitted database schema formatted as pseudo-DDL, which is enriched with table comments, column descriptions, data types, and foreign key relationships, while strictly enforcing access control through a policy to prevent unauthorized data exposure. Furthermore, the orchestrator appends operational guardrails, including explicit SQL safety instructions and dynamic language directives derived from session HTTP headers to enforce the final output language. By providing this multi-faceted state, the orchestrator reduces schema hallucinations and ensures the LLM's reasoning aligns with both the business logic and the structural reality of the underlying database environment.

The generation of SQL is decoupled from the main conversational agent and encapsulated within a dedicated tool interface. This component enforces a programmatic contract: it accepts natural language inputs and outputs structured execution results alongside the generated SQL. To address the challenges of value grounding and structural mapping within complex schemas, our system implements a two-tier RAG approach. The structural layer mitigates syntax errors by injecting Dynamic Few-Shot SQL examples [Brown et al. 2020]. Specifically, it computes vector embeddings for the user's natural language query and matches them against a memory bank of validated queries, selecting up to K examples based on a t cosine similarity threshold. Simultaneously, the semantic layer ensures precise value grounding by extracting candidate entities from the user's input via a LLM call and matching them against actual database contents. To achieve this, the system targets text-based columns within authorized tables, extracting distinct values up to a predefined limit. The entity matching is executed through a multi-stage pipeline: it begins with normalized string comparisons (NFKD) and MinHash [Broder 1997] algorithms for lexical filtering, followed by a deeper semantic refinement using vector

embeddings bounded by a specific threshold T. The resulting dictionary of verified literals is injected directly into the LLM's system prompt with explicit directives to restrict generation to these concrete values.

Recognizing the cognitive conflicts associated with purely generative self-correction models [Hong et al. 2026], our architecture employs a self-repair loop that feeds actual database execution errors back into the prompt. If the database management system returns an execution error, the system initiates a retry mechanism, capped at k iterations. The exact execution error is injected back into the prompt, anchoring the model's refinement process in deterministic technical feedback.

3.2. Evaluated Strategies

In order to improve the efficiency of the Text-to-SQL system, in the scope of both latency and correctness, we performed tests based on databases describing Statement of Operations metrics about companies. In our tests, we evaluated different planning and tool use techniques, isolated and combined. The evaluated planning strategies were ReAct [Yao et al. 2023], ReWOO [Xu et al. 2023], Reflexion [Shinn et al. 2023], Self-Refine [Madaan et al. 2023], Multiplan (a long version and a shorter one) [Wang et al. 2023], and two versions of the SwiftSage architecture [Lin et al. 2023]. For the tool use techniques, we tested the absence of strategies (Default), Dynamic Few-Shot Learning [Brown et al. 2020], and the ToolVerifier [Mekala et al. 2024] framework.

The ReWOO (Reasoning Without Observation) [Xu et al. 2023] technique isolates the plan generation step from the tool calls by first generating a complete plan and then executes all its steps, gathering the results and reasoning about them to construct the final answer. Another type of architecture is that of the self-refinement loop. Both Self-Refine [Madaan et al. 2023] and Reflexion [Shinn et al. 2023] implement a loop of generation and improvement of the answer through critique and constructive feedback, the difference being that Reflexion implements a dialog between a critic node, whose critiques are used as feedback by the generator node to improve the quality of the answer, while Self-Refine has the same agent generate, decide whether the response is satisfactory and apply improvements to the final answer without calling external nodes.

The Multiplan [Wang et al. 2023] method tries to enhance the quality of the agent's response by giving it degrees of freedom to generate different responses and then choosing the most fitting one. This can be done by generating multiple plans varying the LLM's temperature or by generating an initial plan and then using the same agent to create different derived plans from it. In our tests, we also used a short version of the Multiplan strategy, which consists of using only two candidate plans with LLM temperatures of 0 and 1 so as to maximize variability and minimize latency gain. Finally, we tested the SwiftSage architecture [Lin et al. 2023], which emulates the two systems of human reasoning described in [Kahneman 2011]: one is faster and intuitive (mimicked by a weaker LLM, called the Swift node) and one is slower but more deliberative and preciser (mimicked by a more powerful LLM, called the Sage node). Additionally, we tested a version of SwiftSage with specific directives involving field specialties and good practice instructions, to which we refer as the revised version.

The Few-Shot strategy [Brown et al. 2020] is widely known and consists of injecting examples of correct tool calls responding to user questions as well as the reasoning

behind them. In this paper, we evaluated the Dynamic Few-Shot. The second tool use strategy used was the ToolVerifier framework [Mekala et al. 2024], which is implemented by adding a verification step after the generation of the tool calls by the agent and their execution. By validating the type signatures of the tools' arguments, this node prevents syntactically incorrect and semantically inconsistent tool calls from being executed.

3.3. Experimental Details

We conducted an empirical analysis grounded in a real-world enterprise scenario. Specifically, the system was deployed and validated through a case study in a consulting company. The experiments utilized complex relational databases describing Statement of Operations metrics for various companies. The dataset encompasses over 10 consecutive years of historical records for more than 15 distinct corporate entities, structured within a normalized relational schema of 5 interconnected tables, comprising over 35 total columns and approximately 10,000 records (e.g., Companies, Chart of Accounts, Historical Values). Furthermore, the curated dataset, comprising a total of 240 natural language queries paired with their gold-standard SQL scripts, was partitioned into two sets to prevent data leakage: a 'Validated Query Bank' containing 20 pairs utilized exclusively as a retrieval pool for Dynamic Few-Shot prompting, and an isolated 'Evaluation Pool' of 220 queries that served as the blind test set to assess execution accuracy and intent alignment. To characterize our benchmark, these 220 queries were categorized by structural and functional complexity: 20% require simple aggregations and basic filtering (Low Complexity), 30% involve multiple JOINs and cross-temporal grouping (Medium Complexity), and 50% demand advanced analytical constructs such as Common Table Expressions (CTEs) and Window Functions (High Complexity).

This environment provided a realistic testbed, characterized by complex schemas, domain-specific financial terminology, and the strict data governance requirements. While standardized Text-to-SQL benchmarks excel at evaluating structural mapping and schema linking, they lack the domain-specific analytical complexity required in enterprise environments. Corporate data querying extends beyond retrieving explicit columns; it demands pragmatic reasoning to infer implicit mathematical formulas, cross-temporal aggregations, and advanced window functions. As illustrated in Table 1, our proprietary dataset encapsulates these real-world cognitive challenges.

For the language modeling components, the orchestrator agent utilized the Gemini 3 Flash model to interpret user intent, manage dynamic context, and decide on tool invocations. The specialized generative tasks were also delegated to the Gemini 3 Flash model. For Dynamic Few-Shot Learning, empirical testing established that retrieving $K = 3$ contextual examples, using a cosine similarity threshold of $t = 0.6$, yielded the best results. While Gemini 3 Flash served as the model for general agent orchestration and low-latency SQL generation, strategies demanding deeper analysis employed more robust setups. Specifically, the Multiplan technique was configured to evaluate either two or four distinct plans to assess the impact of reasoning diversity on overall accuracy. For the dual-system SwiftSage architecture, task allocation followed fast and slow thinking paradigms: Gemini 3 Pro powered the deliberative "Sage"node, and Gemini 3 Flash handled the intuitive "Swift"node. Similarly, in the Reflexion strategy, Gemini 3 Pro acted as the critic, leveraging its advanced reasoning to provide the introspective feedback necessary to refine the base model's outputs.

Tabela 1. Analytical Complexity of Enterprise NL2SQL Tasks

Q1: Calculate % CAGR of Gross Revenue (2020-2025).
Analytical Challenge: Requires cross-temporal aggregation and the inference of implicit financial formulas not stated in the prompt.

```
SELECT ROUND((POWER(MAX(v25)/MAX(v20), 1/5.0)-1)*100, 2)
AS cagr_pct...
```

Q2: Calculate the Degree of Operating Leverage (2024-2025).
Analytical Challenge: Demands multi-step Common Table Expressions (CTEs) to isolate metrics before calculating dynamic ratios.

```
SELECT ROUND((var_ebit / NULLIF(var_rec_liq, 0))::numeric, 2)
AS op_leverage...
```

Q3: Build a Pareto curve for Net Income in 2025.
Analytical Challenge: Relies on advanced PostgreSQL window functions to compute cumulative distributions dynamically.

```
SELECT SUM(lucro) OVER (ORDER BY lucro DESC ROWS UNBOUNDED
PRECEDING AND CURRENT ROW) AS acum...
```

Note: Standard structural syntax (e.g., JOINs, WHERE clauses) is omitted to highlight the mathematical and declarative reasoning gap between enterprise queries and standard benchmarks.

The Text-to-SQL generation was executed with a low temperature setting of 0.1. This aimed to minimize variance and reduce the likelihood of hallucinations. Furthermore, before any query was sent to the database, it was processed using the sqlparse library [Albrecht 2009] to guarantee readability and lexical normalization. Robust error handling and iterative refinement were central to the experimental setup. We configured the self-repair loop with a maximum of three retry attempts for the Text-to-SQL module. Finally, the automated evaluation of the generated SQL queries prioritized semantic accuracy and analytical validity over rigid syntactic comparisons. To achieve this, we employed the DeepEval framework [Ip 2023], specifically leveraging the G-Eval [Liu et al. 2023] metric for *SQL Text2SQL Correctness* with Gemini 3 Pro. This approach evaluated whether the generated query accurately captured the user's underlying analytical intent—such as applying the correct filters, aggregations, joins, and columns—rather than demanding an exact textual match against a human-written ground-truth script.

This automated assessment, alongside metrics for execution success and processing latency, provided an evaluation of the architecture. To overcome the limitations of binary evaluation methodologies, we incorporated a Relaxed Execution Accuracy (Relaxed EX) metric [Pinna et al. 2025]. This approach provides a granular evaluation by quantifying the degree of difference between the result tables. By calculating the normalized edit distance between the columns of the generated table and the gold-standard reference table, the metric generates a similarity score ranging from 0 to 1 [Pinna et al. 2025].

To establish a rigorous evaluation baseline, a custom test dataset was constructed for this study. This dataset comprises natural language queries paired with their corresponding gold-standard SQL scripts and expected textual answers, alongside a framework for the human evaluation of the generated outputs. The curation and assessment of this dataset were conducted by domain experts from the case study company. These professionals underwent specific training to guarantee strict adherence to corporate standardization guidelines, data governance protocols, and SQL best practices, thereby ensuring a reliable

valid benchmark for our empirical analysis.

Complementing the automated metrics, we implemented an human evaluation protocol to assess the correctness of the system's final outputs. For every query in the test dataset, two independent, trained domain experts evaluated the system's performance, assigning a binary score to indicate whether the generated response accurately satisfied the user's intent. To make these informed judgments, the evaluators cross-referenced the original natural language question, the gold-standard SQL, the expected gold response, the model-generated SQL, and the final textual output. Furthermore, to ensure the consistency and reliability of this manual assessment, inter-rater agreement was measured using Cohen's Kappa, yielding a score of 0.82 prior to the final scoring.

4. Results and Discussions

The experimental evaluation of our proposed Text-to-SQL architecture aimed to isolate the impact of different cognitive planning strategies and tool-use techniques in navigating complex enterprise databases. Table 2 presents a comprehensive comparison of these paradigms across four key metrics: Latency, Human Evaluation, Relaxed Execution Accuracy (Relaxed EX), and SQL Correctness. By analyzing these dimensions, we can effectively delineate the trade-offs between syntactic precision, pragmatic intent alignment, and computational cost.

Tabela 2. Experimental Results of Tool Use and Planning Techniques

Tool Use Technique	Planning Technique	Latency (s)	Human Eval. (%)	Relaxed EX (%)	SQL Correct. (%)
Default	ReAct	94.17 ± 1.25	78.57 ± 1.43	75.00 ± 1.05	72.11 ± 2.07
Tool Verifier	ReAct	112.69 ± 2.31	78.57 ± 1.51	75.00 ± 1.12	75.56 ± 1.92
Few Shot	ReAct	107.69 ± 1.42	82.14 ± 1.20	$\mathbf{78.57 \pm 1.02}$	73.58 ± 1.88
Default	ReWOO	90.87 ± 1.94	67.86 ± 2.03	75.00 ± 1.77	78.02 ± 2.91
Few Shot	Reflexion	200.95 ± 2.51	75.00 ± 1.92	75.00 ± 1.63	82.44 ± 1.73
Default	Self Refine	168.05 ± 1.59	67.86 ± 1.70	75.00 ± 1.30	77.96 ± 1.52
Default	Multiplan	127.86 ± 1.67	85.71 ± 1.08	75.00 ± 1.36	82.53 ± 2.04
Few Shot	Multiplan	141.35 ± 1.02	$\mathbf{89.29 \pm 2.93}$	75.00 ± 1.68	81.30 ± 2.11
Few Shot	Multiplan (Short)	130.19 ± 2.84	$\mathbf{89.29 \pm 2.57}$	$\mathbf{78.57 \pm 1.04}$	79.43 ± 1.72
Few Shot	SwiftSage	$\mathbf{84.34 \pm 2.37}$	82.14 ± 1.22	71.43 ± 2.90	$\mathbf{83.59 \pm 1.44}$
Few Shot	SwiftSage (Revised)	101.04 ± 1.92	$\mathbf{89.29 \pm 1.72}$	75.00 ± 1.90	80.03 ± 1.66

Starting with the ReAct paradigm, the baseline implementation utilizing Default tool use achieved a Latency of 94.17 seconds, a Human Evaluation score of 78.57%, and a Relaxed EX of 75.00%. When augmenting ReAct with the ToolVerifier framework, we observed a marginal increase in SQL Correctness to 75.56%, but this came at the cost of increased Latency (112.69 seconds) without yielding tangible gains in execution or human alignment. Conversely, integrating Dynamic Few-Shot learning with ReAct elevated the Relaxed EX to 78.57% and improved Human Evaluation to 82.14%. This indicates that injecting contextually relevant structural examples significantly aids the model in pragmatic value grounding, outperforming strictly technical verification steps.

Analyzing the decoupled planning architecture, the ReWOO strategy combined with Few-Shot learning demonstrated a competitive Latency of 90.87 seconds and a rela-

tively strong SQL Correctness of 78.02%. However, this configuration suffered a degradation in both Human Evaluation and Relaxed EX. This suggests that while completely separating strategic plan generation from tool execution helps the model adhere to rigid SQL syntax, it impairs the agent's ability to dynamically adapt to the schema and fulfill the user's actual analytical intent.

Our findings regarding iterative self-correction frameworks, specifically Reflexion and Self-Refine, provide crucial empirical evidence of the cognitive conflicts introduced in agentic Text-to-SQL workflows. Reflexion achieved a high SQL Correctness of 82.44%, but imposed a prohibitive Latency of 200.95 seconds. Self-Refine exhibited similar operational hurdles, with a Latency of 168.05 seconds and a sharp drop in Human Evaluation to 67.86%. These heavy computational overheads did not translate into proportional execution gains; in fact, the Relaxed EX for both Reflexion and Self-Refine remained stagnant at 75.00%. This behavior corroborates our hypothesis that when an LLM attempts to self-correct based on its own internal reasoning rather than deterministic external feedback, it frequently generates false-positive error detections during self-evaluation, degrading previously valid logic [Hong et al. 2026]. The redundancy of internal critique loops often leads the agent to reinterpret the user's intent incorrectly, highlighting why our architecture's reliance on database-driven self-repair is a more stable approach.

In contrast, the Multiplan strategies demonstrated exceptional performance in aligning generated queries with pragmatic user intent. The Default Multiplan configuration yielded a strong SQL Correctness of 82.53% alongside a Human Evaluation of 85.71%. When augmented with Few-Shot examples, the standard Multiplan reached the highest peak Human Evaluation score of 89.29% , indicating that generating and evaluating multiple execution paths significantly reduces hallucinations. To mitigate the inherent latency overhead of generating multiple extensive plans (which peaked at 141.35 seconds for Few-Shot Multiplan), we tested an ablated version: Multiplan (Short). This abbreviated version remarkably maintained the peak Human Evaluation score of 89.29% and achieved the highest overall Relaxed EX of 78.57%, while reducing Latency to 130.19 seconds. This ablation demonstrates that restricting the generation to a smaller, highly variable set of candidate plans effectively optimizes the trade-off between generative diversity and response speed.

The most compelling results for enterprise deployment emerged from the dual-system reasoning architectures. The base SwiftSage model paired with Dynamic Few-Shot learning achieved the lowest overall Latency (84.34 seconds) and the highest SQL Correctness (83.59%) among all tested configurations. This suggests the claim that emulating human cognitive blueprints—dynamically routing queries to either a fast intuitive node or a deliberative analytical node —provides an optimal framework for navigating strict syntactic constraints efficiently.

Furthermore, the SwiftSage (Revised) configuration, which incorporated specific directives involving domain specialties and best practice instructions, successfully bridged the gap between raw syntactic precision and pragmatic alignment. While its Latency slightly increased to 101.04 seconds compared to the base model, it secured the top-tier Human Evaluation score of 89.29% and maintained a solid SQL Correctness of 80.03%.

Ultimately, these empirical results map out evidence-based guidelines for desig-

ning reliable Text-to-SQL agents. Traditional reactive loops and heavy internal self-refinement methods introduce either execution instability or unacceptable latency in complex corporate environments. Conversely, strategies that decouple cognitive planning through targeted candidate paths (Multiplan Short) or utilize dual-reasoning routing (SwiftSage Revised) consistently deliver the highest operational reliability. By offloading error correction to deterministic DBMS feedback rather than LLM introspection, and by structuring the reasoning process through frameworks like SwiftSage, our proposed architecture effectively mitigates cognitive conflicts. This synthesis of structured planning, dynamic two-tier RAG context, and programmatic guardrails successfully transitions Text-to-SQL from an isolated generative task into a robust multi-agent workflow.

Despite the promising empirical results regarding dual-system reasoning and database-driven self-repair, this study presents certain limitations that warrant consideration. First, the experimental evaluation was conducted entirely on a proprietary, domain-specific dataset centered around financial Statement of Operations metrics within a corporate consulting environment. Consequently, the absence of evaluation on standardized, publicly available Text-to-SQL benchmarks, such as Spider or BIRD, limits the verifiable generalizability of the proposed architecture across diverse and open-domain schemas. Second, the system's performance and the observed cognitive conflicts were measured using a single family of Large Language Models (specifically the Gemini 3 series); future research must evaluate whether these planning paradigms exhibit similar efficacy and behavioral patterns when deployed with other frontier and open-weight models. Finally, while this work provides a rigorous analysis of computational overhead through the lens of latency, it omits a comprehensive assessment of the financial costs associated with token consumption. Given that multi-agent frameworks and multi-plan generative strategies inherently demand higher token throughput, a detailed cost-benefit analysis remains essential to fully validate their viability for high-volume enterprise deployments.

5. Conclusions

In conclusion, this study proposed and evaluated a multi-agent Text-to-SQL architecture tailored for the strict demands of complex enterprise environments. By replacing purely generative self-correction with a database-driven self-repair loop and implementing a two-tier RAG approach, the system mitigates the cognitive conflicts and latency overheads inherent to introspective models like Reflexion and Self-Refine. Our empirical findings suggest that dual-system reasoning architectures, particularly the SwiftSage framework augmented with Dynamic Few-Shot learning, provide a favorable balance of low latency, high syntactic precision, and pragmatic intent alignment within the evaluated context. Ultimately, this research highlights that decoupling cognitive planning from deterministic technical execution appears to be a critical factor for transitioning Text-to-SQL from an isolated generative task toward a more reliable corporate tool within the evaluated context.

Building upon these foundational insights, future work should prioritize validating the proposed architecture across benchmarks to confirm its generalizability beyond proprietary schemas. Furthermore, investigating the efficacy of these planning paradigms using a diverse array of open-weight and alternative frontier models will be crucial for understanding how different underlying cognitive capabilities impact structural mapping. Finally, to address the computational and financial overhead associated with generating multiple candidate paths or orchestrating heavy dual-reasoning loops, subsequent research

should explore dynamic routing mechanisms. By developing lightweight classifiers that assess query complexity in real-time, systems could adaptively trigger resource-intensive planning strategies only when strictly necessary.

Referências

Albrecht, A. (2009). sqlparse: A non-validating sql parser for python.

Baptista, J. V. R. and Garcia, L. P. F. (2025). Graphrag on electronic health record: A knowledge graph-enhanced rag approach for healthcare information access. In *Brazilian Conference on Intelligent Systems (BRACIS)*, volume 35, pages 615–630, Porto Alegre. Sociedade Brasileira de Computação.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

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.

Chaturvedi, S., Chadha, A., and Bindschaedler, L. (2024). Sql-of-thought: Multi-agentic text-to-sql with guided error correction. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Chen, Z., Chen, S., White, M., Mooney, R., Payani, A., Srinivasa, J., Su, Y., and Sun, H. (2023). Text-to-sql error correction with language models of code. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Eckmann, M. et al. (2025). Hlr-sql: Human-like reasoning for text-to-sql with the human in the loop. *Information Systems*.

Hong, Z., Chen, H., Yuan, Z., Zhang, Q., Zhuang, L., Liao, Q., Huang, F., Song, Y., and Huang, X. (2026). Errorllm: Modeling sql errors for text-to-sql refinement. arXiv preprint arXiv:2603.03742.

Ip, J. (2023). Deepeval: The llm evaluation framework. Confident AI.

Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux.

Kamoi, R., Zhang, Y., Zhang, N., Han, J., and Zhang, R. (2024). When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Katsogiannis-Meimarakis, G. and Koutrika, G. (2023). A survey on deep learning approaches for text-to-sql. *The VLDB Journal*.

Lin, B. Y., Fu, Y., Yang, K., Ammanabrolu, P., Brahman, F., Huang, S., Bhagavatula, C., Choi, Y., and Ren, X. (2023). Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Liu, X., Shen, S., Li, B., Ma, P., Jiang, R., Zhang, Y., Fan, J., Li, G., Tang, N., and Luo, Y. (2025). A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*.

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. (2023). G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. (2023). Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Martinelli, I. and Ponti, M. A. (2025). Towards secure retrieval-augmented generation: Preventing llm data leaks with rbac. In *Brazilian Conference on Intelligent Systems (BRACIS)*, volume 35, pages 500–515, Porto Alegre. Sociedade Brasileira de Computação.

Mekala, D., Weston, J., Lanchantin, J., Raileanu, R., Lomeli, M., Shang, J., and Dwivedi-Yu, J. (2024). Toolverifier: Generalization to new tools via self-verification. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Pinna, G., Perezhohin, Y., Manzoni, L., et al. (2025). Redefining text-to-sql metrics by incorporating semantic and structural similarity. *Scientific Reports*.

Pourreza, M., Li, H., Sun, R., Chung, Y., Talaei, S., Kakkar, G. T., Gan, Y., Saberi, A., Ozcan, F., and Arik, S. (2025). Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. In *International Conference on Learning Representations (ICLR) Workshop*.

Shi, L., Tang, Z., Zhang, N., Zhang, X., and Yang, Z. (2025). A survey on employing large language models for text-to-sql tasks. *ACM Computing Surveys*.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

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, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.

Xu, B., Peng, Z., Lei, B., Mukherjee, S., Liu, Y., and Xu, D. (2023). Rewoo: Decoupling reasoning from observations for efficient augmented language models. arXiv preprint arXiv:2305.18323.

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 *International Conference on Learning Representations*.

Zhang, X., Liu, Y., Wang, Z., et al. (2026). Text-to-graph query using semantic subgraph retrieval. *Knowledge-Based Systems*.