CTU Journal of Innovation and Sustainable Development
ISSN 2588-1418 | e-ISSN 2815-6412

DOI:10.22144/ctujoisd.2025.058

Enhancing text-to-SQL capabilities of small language models via schema context enrichment and self-correction

Le Gia Kiet, Le Quoc Khanh, Nguyen Minh Nhut, and Nguyen Dinh Thuan*

Faculty of Information Systems, University of Information Technology, Vietnam National University Ho Chi Minh City, Viet Nam

**Corresponding author (thuannd@uit.edu.vn)*

Article info.

Received 8 Jul 2025
Revised 15 Aug 2025
Accepted 5 Oct 2025

Keywords

Database schema context enrichment, graph clustering, natural language processing, open-source models, small language models (SLMs), text-to-SQL

ABSTRACT

Translating natural language into SQL is essential for intuitive database access, yet open-source small language models (SLMs) still lag behind larger systems when faced with complex schemas and tight context windows. This paper introduces a two-phase workflow designed to enhance the Text-to-SQL capabilities of SLMs. Phase 1 (offline) transforms the database schema into a graph, partitions it with Louvain community detection, and enriches each component in a cluster with metadata, relationships, and sample rows. Phase 2 (at runtime) selects the relevant tables, generates SQL queries, and iteratively refines the SQL through an execution-driven feedback loop until the query executes successfully. Evaluated on the Spider test set, our pipeline raises Qwen-2.5-Coder-14B to 86.2% Execution Accuracy (EX), surpassing its zero-shot baseline and outperforming all contemporary SLM + ICL approaches and narrowing the gap to GPT-4-based systems all while running on consumer-grade hardware. Ablation studies confirm that both schema enrichment and self-correction contribute significantly to the improvement. The study concludes that this workflow provides a practical methodology for deploying resource-efficient open-source SLMs in Text-to-SQL applications, effectively mitigating common challenges. An open-source implementation is released to support further research.

1. INTRODUCTION

The rapid advancement of Large Language Models (LLMs) has significantly transformed natural language processing, demonstrating impressive capabilities across various tasks. In the realm of database interaction, these models have greatly enhanced Text-to-SQL systems, which translate natural language questions into executable SQL queries (Gao et al., 2023; Pourreza & Rafiei, 2023; Li et al., 2025; Wang et al., 2025). State-of-the-art proprietary models like GPT-4o demonstrate this potential by achieving high accuracy on complex benchmarks (Hong et al., 2025). However, the substantial computational resources required by these large models often limit their accessibility and practicality for widespread deployment. This has spurred growing interest in Small Language Models (SLMs), which offer a more resource-efficient alternative, balancing performance with lower costs and deployment flexibility, making them attractive candidates for specific applications like Text-to-SQL interfaces.

Despite the promise of SLMs, applying open-source variants to complex, real-world Text-to-SQL tasks remains challenging. Practical database schemas often dwarf the complexity found in benchmarks

like Spider, leading to significant performance degradation, particularly for smaller models. Furthermore, open-source SLMs generally lag behind proprietary counterparts like GPT-4o in handling intricate queries and large schemas (Chen et al., 2024), often due to their scale and inherent limitations.

1.1. Core challenges

Although open-source small language models (SLMs) have made notable strides, their deployment in practical Text-to-SQL systems is still hampered by four interlocking challenges that jointly constrain schema understanding, context retention, and query correctness.

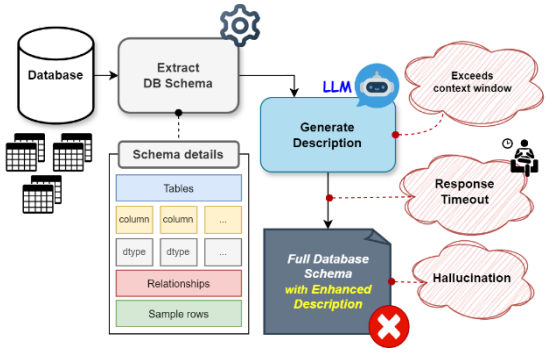

Figure 1. Challenges in vanilla database schema context enrichment

– *Understanding Complex Schemas:* Grasping the semantics of large, real-world database schemas, SLMs struggle to comprehend relationships in large/ambiguous schemas, leading to mapping errors and severely degrading SQL quality.

– *Context-Window Limitations:* Finite context windows restrict the amount of information that can be processed. Including complete details for expansive schemas is often infeasible, a problem exacerbated for SLMs which may have smaller windows and weaker long-input comprehension, forcing a trade-off between context sufficiency and overload.

– *Accuracy and Hallucination:* Generating incorrect or non-executable SQL is a common issue. SLMs may hallucinate tables/columns or create flawed logic, particularly for complex operations like multi-table joins or nested queries, impacting the reliability of the output.

– *Performance Gap and Self-Correction Needs:* A noticeable performance gap exists compared to leading proprietary models, with SLMs tending to produce more errors. This necessitates robust self-correction mechanisms that leverage execution feedback, as basic error messages often prove insufficient for these models.

Figure 1 illustrates how these general obstacles manifest concretely in the schema-description enrichment step, where the system extracts metadata from a database and uses an LLM to generate enhanced descriptions.

1.2. Novel two-phase workflow approach

To overcome the foregoing obstacles, we introduce a two-phase workflow that lifts the Text-to-SQL performance of open-source SLMs without exceeding consumer-grade resources.

1.2.1. Database schema context enrichment

Instead of truncating schemas, we model the database as a graph (tables as nodes, foreign keys as edges) and apply Louvain community detection (Banda & Motik, 2020) to cluster tables into semantically coherent groups. This reduces complexity by processing related tables together. Each cluster's context is enriched with metadata, relationships, and sample data, enabling the SLM to infer semantics and reduce hallucinations.

1.2.2. SQL generation and self-correction

For large databases, relevant schema are first selected based on the natural language query. The SLM then generates an initial SQL query using the enriched context. Critically, a rule-based validation step is applied to ensure the generated SQL is syntactically correct. Then, an execution-based self-correction loop iteratively refines the query: it's executed against the database, and any error messages are fed back to the SLM for revision until the query runs successfully or a retry limit is met, grounding generation in real-world validity.

This two-stage design addresses key bottlenecks such as schema understanding, context limits, and generation correctness providing a scalable framework for deploying open-source SLMs in realistic Text-to-SQL scenarios.

1.3. Key contributions

Our main contributions center on enhancing Text-to-SQL capabilities for open-source Small Language Models (SLMs). **(1)** We introduce a novel two-phase workflow specifically tailored to address the challenges SLMs face with schema complexity, context limitations, and query accuracy. **(2)** This workflow includes a schema enrichment strategy that employs graph modeling and Louvain clustering to create semantically coherent table

groups, enabling more efficient and context-aware prompting. Furthermore **(3)** we present a multi-step query generation framework incorporating rule-based validation and, crucially, an execution-driven self-correction loop that utilizes real database feedback to iteratively refine SQL accuracy. To promote transparency, reproducibility, and broader adoption, **(4)** we also provide an *open-source implementation* compatible with diverse SLM backends.

1.4. Related work

Text-to-SQL research evolved from early systems to Pre-trained Language Models like BERT and T5 (Wong et al., 2024), which improved translation but often required significant fine-tuning or struggled with complex schemas (Qi et al., 2022; Li et al., 2023). Subsequently, Large Language Models (LLMs) like GPT, Gemini, and Claude advanced the field using In-Context Learning (ICL) or Fine-Tuning (FT) (Hong et al., 2025). Alongside these advancements, attention has increasingly turned to *open-source Small Language Models (SLMs)* (e.g., Llama, Phi, Qwen (Hui et al., 2024)). While offering transparency and efficiency benefits, these SLMs face pronounced challenges with large schemas and complex queries compared to proprietary LLMs, often due to inherent scale limitations affecting context handling and comprehension (Li et al., 2024; Mohammadjafari et al., 2025).

A key challenge remains schema understanding and linking, where language models, *particularly open-source SLMs*, can falter with large or ambiguous schemas (Gao & Luo, 2025). Addressing this often involves techniques like schema filtering, graph-based representations (GNNs, clustering) (Cai et al., 2021) for efficient structuring, context enrichment, bidirectional linking (Cao et al., 2024), schema reduction, and using Retrieval-Augmented Generation (RAG) to dynamically provide relevant schema context. Such methods are especially pertinent for SLMs operating under tighter constraints. Our graph-based clustering aligns with these efforts, aiming to manage schema complexity within the typical context capabilities of SLMs.

Ensuring SQL accuracy is another critical hurdle, as language models may generate incorrect SQL or hallucinate elements. This demands robust solutions, especially when working with SLMs. Relevant approaches include query decomposition (e.g., DIN-SQL; Pourreza & Rafiei, 2023, and DTS-SQL which specifically targets SLMs; Pourreza & Rafiei, 2024), intermediate representations (NatSQL; Gan et al., 2021), Chain-of-Thought prompting (Tai et al., 2023), and self-correction mechanisms. Techniques leveraging execution-based feedback (MAC-SQL; Wang et al., 2025) or critic models (SQLCritic; J. Chen et al., 2025) are particularly valuable for refining potentially inaccurate outputs from these SLMs, inspiring our own execution-driven correction loop.

Standard evaluation uses benchmarks like Spider (Yu et al., 2019) with metrics like Execution Accuracy (EX) and Exact Matching (EM). The recognized performance gap for open-source SLMs on such complex tasks provides strong motivation for our two-phase workflow, which combines graph-based schema enrichment and execution-driven self-correction specifically to enhance their Text-to-SQL performance.

2. MATERIALS AND METHOD

This section outlines our proposed two-phase methodology aimed at improving the accuracy of smaller open-source LLMs (less than 14B parameters) on complex Text-to-SQL tasks. The workflow begins with Phase I: Database Schema Context Enrichment, an offline process that converts intricate database schemas into more manageable and semantically richer representations tailored to LLM context limits, utilizing graph-based clustering (as depicted in Figure 2). Next, Phase II: SQL Generation and Self-Correction takes place at query time, utilizing the enriched schema context to translate natural language questions into SQL queries and incorporating an execution-based self-correction loop to enhance accuracy (illustrated in Figure 3). This structured approach methodically tackles challenges such as schema complexity, context window limitations, and query inaccuracies inherent in smaller models.

2.1. Materials

2.1.1. Models

To evaluate our schema enrichment and SQL generation pipeline, we compare two open-source models with distinct design emphases:

Qwen2.5-Coder:14B (Hui et al., 2024) - A code-specialized variant of Qwen2.5 model and further refined for programming tasks. Its strengths in syntax, schema comprehension, and code reasoning make it a good candidate for SQL generation.

Phi-4:14B - A generalist model focused on reasoning, diverse training data underpins strong logical across domains.

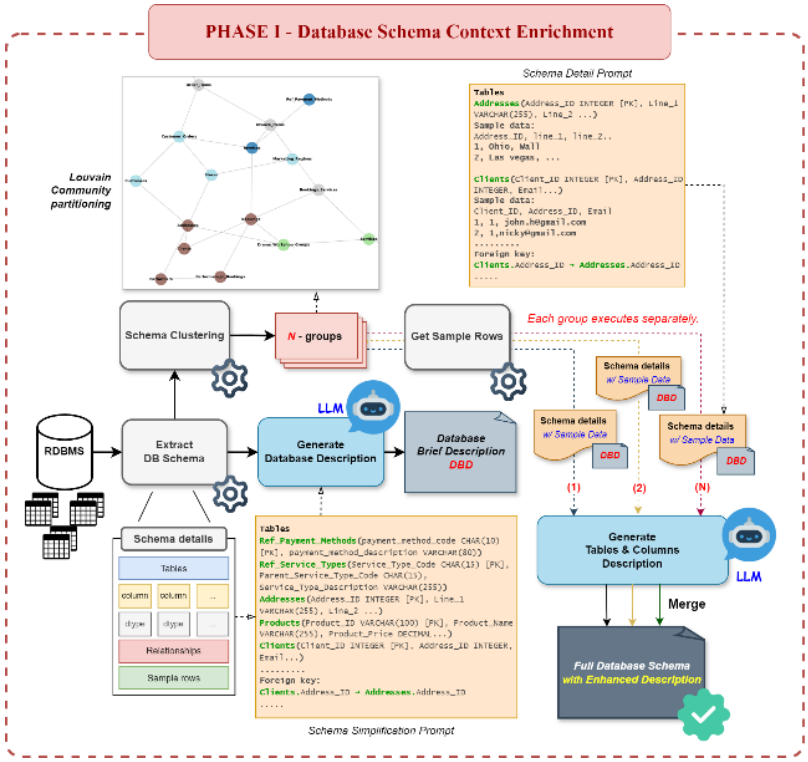

Figure 2. Overview of the proposed Database Schema Context Enrichment methodology

Both models are deployed in quantized format Q4_K_M GGUF format to enable efficient inference on consumer-grade hardware while maintaining high performance. For the experiments, we use a machine equipped with an RTX 4060 Ti GPU with 16GB of VRAM, representing typical hardware constraints faced by end-users. This comparison reveals whether a code-focused architecture or a broadly trained generalist better leverages enriched schema context for accurate SQL synthesis.

2.1.2. Datasets

We evaluate on the Spider dataset (Yu et al., 2019), a large-scale, cross-domain Text-to-SQL benchmark comprising *10,181 questions* and *5,693 SQL queries* over *200 databases* in *138 domains*. Its complex queries (e.g., multi-table joins, nested subqueries) make it a standard testbed; we report results on the official development and test splits.

2.2. Method

2.2.1. Database schema context enrichment

Database Schema Graph Representation - The relational database schema is formally modeled as an undirected graph

$$G = (V, E)$$

where the set of vertices V represents the tables within the database, and the set of edges E represents the foreign key relationships between tables.

These edges are extracted from the *relations* field within column metadata, with duplicates removed for computational efficiency. The resulting graph structure effectively captures the inherent connectivity of the database schema.

$$E = \{ (u, v) \mid u, v \in V \wedge \exists\, ForeignKey(u, v) \}$$

Louvain-based community detection - To effectively manage complex database schemas, we apply the Louvain community detection algorithm (Banda & Motik, 2020) to the schema graph. This algorithm partitions the graph by maximizing the modularity metric Q:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

where A_{ij} represents the edge between nodes i and j, k_i is the sum of edges connected to node i, c_i is

the community of node i, and δ is the *Kronecker delta* function. As an optimization, for schemas where $|V| < 5$, we simply define a single cluster containing all tables. For larger schemas, we construct an appropriate graph representation and apply the community detection algorithm with a parameter $resolution = 2.5$ to control community granularity.

The algorithm yields a partition $\mathcal{P} = \{C_1, C_2, \dots, C_k\}$ of tables into disjoint clusters. These clusters are then sorted by size in descending order, with tables within each cluster alphabetically ordered by their identifiers for consistent presentation.

Overview database schema context generation - Before the cluster-level enrichment phase, the system first compresses the entire schema G into a simple prompt representation to save tokens, yielding

$$P_{\text{simple}} = SIMPLE(G)$$

From this compact prompt P_{simple}, the SLM is instructed to generate a global description of the database, formalised by the function

$$\mathcal{E}_{\text{global}}(G) = \{ D_{\text{db}}, T_{\text{desc}} \}$$

More concretely,

- D_{db} = a concise description of the whole database
- $T_{\text{desc}} = \left\{ \left(T_j, D_{T_j}^{\text{short}} \right) \middle| T_j \in G \right\}$ (table description)

The resulting set $\mathcal{E}_{\text{global}}(G)$ acts as a high-level "mental map" that underpins the subsequent *cluster-specific context enrichment* performed for every cluster C_i.

Cluster-specific context generation - For each identified cluster C_i, we generate comprehensive contextual information to provide the LLM with sufficient understanding of the related tables. The enrichment process can be formalized as a function $\mathcal{E}$, that generates structured context for each cluster:

$$\mathcal{E}(C_i) = \{D_{db}, T_{meta}, Col_{info}, R_{fk}, S_{data}\}$$

where:

- D_{db} = *Database description*
- $T_{meta} = \left\{ \left(T_j, D_{T_j} \right) \middle| T_j \in C_i \right\}$
- $Col_{info} = \left\{ \left(col_{jk}, type_{jk}, key_{jk}, D_{col_{jk}} \right) \middle| col_{jk} \in T_j,\ T_j \in C_i \right\}$
- $R_{fk} = \{(T_a, col_a, T_b, col_b) \mid T_a, T_b \in C_i, (T_a, T_b) \in E\}$
- $S_{data} = \left\{ \left(T_j, \{r_1, \dots, r_m\} \right) \middle| T_j \in C_i \right\}$

The enriched representation includes database/table descriptions, detailed column metadata (names, types, keys), inter-table relationships (within and across clusters), and sample data rows to clarify semantics.

The implementation follows a workflow pattern with error handling and retry mechanisms. For each cluster, the process attempts up to three retries if the initial generation fails. After generation, the descriptions are mapped back to the original schema, updating only fields that were previously empty or contained placeholder values.

$$\tau_i = \min\{ j \in \{1,2,3\} : E_{j(C_i)} \neq \emptyset \},$$

$$E_{final}(C_i) = \begin{cases} E_{\tau_i}(C_i), & \tau_i \ exists, \\ \emptyset & , otherwise, \end{cases}$$

This approach preserves existing high-quality metadata while enriching incomplete elements. The enrichment process overcomes limitations of purely structural clustering by providing localized context that integrates structural details, semantic descriptions, and concrete data examples. The workflow tracks metrics including success rates and processing times to measure effectiveness.

2.2.2. SQL generation and self-correction

This phase executes at query time, taking a user's natural language question (NLQ) and the enriched schema representations from Phase I. The workflow implements a multi-step process with selective table filtering, SQL generation, validation, execution, and error correction. The detailed process is shown in Figure 3.

Relevant table selection - When a database contains many tables, we first narrow down to only those likely needed to answer the user's query. Formally, we define:

$$\mathcal{S}:\ Q \times \{T_1, T_2, \dots, T_n\} \longrightarrow \{T_{r_1}, T_{r_2}, \dots, T_{r_m}\},$$

Where Q is the natural language question and $\{T_{r_j}\}$ is the subset of tables deemed relevant.

Concretely:

$$\mathcal{S}(Q, \{T_i\}_{i=1}^{n}) = \begin{cases} \{T_i\}_{i=1}^{n} & , n \leq \tau, \\ \mathcal{S}_{SLM}(Q, \{T_i\}_{i=1}^{n}) & , n > \tau, \end{cases}$$

With threshold τ (e.g. $\tau = 3$). when $n > \tau$, we call:

$$\mathcal{S}_{SLM}(Q, \{T_i\}) = \text{SLM}(P_{\text{retrieval}}),$$

where the prompt $P_{\text{retrieval}}$ contains:

- The user's query Q,
- A list of all *table names* T_i alongside their brief *descriptions*,
- An instruction to "return the full set of all tables that might be relevant to answering Q."

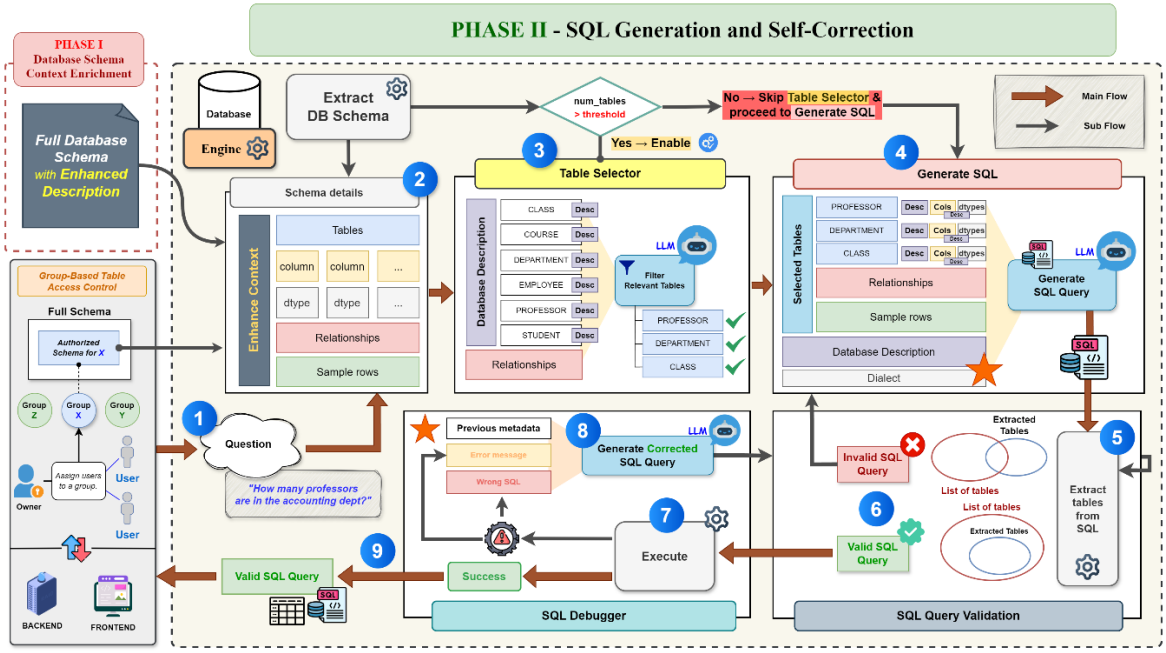

Figure 3. Comprehensive framework - Schema Context Enrichment (Phase I, Figure.2) and SQL Generation & Self-Correction (Phase II) with table access control, LLM-based table selection, rule-based validation, and execution-driven correction

SQL generation and validity checking - Once the relevant table set has been determined, the system proceeds to generate and validate the first SQL statement using the prepared context. The workflow comprises four main stages:

a. Query generation

A prompt $P_{\text{generation}}$ is submitted to the SLM to obtain the initial SQL query:

$$SQL_{\text{initial}} = \text{SLM}(P_{\text{generation}}),$$

$$P_{\text{generation}} = \{I, S_C, Q, dialect\},$$

where

- I - An instruction that *forces the model to output only a single SQL statement.*
- S_C - Schema Context Serialisation (Restricted to Selected Tables)
- Q - the user's normalised English question
- $dialect$ - the target SQL query syntax (e.g. PostgreSQL or MySQL)

b. SQL extraction and syntactic check

From the SLM's response, all non-SQL content (explanations, formatting, etc.) is stripped, leaving a *clean SQL string*. A lightweight parser is then invoked to ensure that the statement is syntactically valid.

c. Table-set validation

The set of tables referenced by the query excluding intermediate CTEs is denoted $\text{Tables}(SQL)$. It is compared against the

permitted set T identified in *Relevant Table Selection* step.

$$V(SQL,T) = \begin{cases} True\ , & if\ \text{Tables}(SQL) \subseteq T, \\ False, & otherwise \end{cases}$$

d. Automatic context expansion (if necessary) If $V(SQL,T) = \text{True}$ but the query references tables that were *not* included in the original context, the system automatically enlarges the relevant-table set, refreshes S_C, and re-invokes the generation phase with the updated prompt. This guarantees that the final query is both syntactically correct and contextually coherent.

Combining *both* syntactic validation and table-set consistency checking allows the system to exert strict quality control over the generated query, thereby minimizing errors caused by incorrect table references or by requests that exceed the context provided to the SLM.

Execution and self-correction loop - After passing static validation, the candidate SQL statement is executed against the database. If no error occurs ($error = \emptyset$), execution terminates, and the result is returned. Otherwise, the system enters the self-correction loop described in *Algorithm 1*. In each iteration, the prompt $P_{\text{generation}}$ supplies the model with the database schema, the original user query Q, the most recent SQL attempt, the error message, and the SQL dialect. By leveraging direct execution feedback, this loop effectively resolves such as syntax errors, schema mismatches, and semantic issues until either a valid query is produced or reach limit retry is reached.

Algorithm 1: SQL Self-Correction Loop

$SQL_{\text{current}} \leftarrow SQL_{\text{initial}}$
$retry \leftarrow 0$
while $retry < max\_retries$ **do**
  $(result, error) \leftarrow \text{Execute}(SQL_{\text{current}}, DB)$
  if $error = \emptyset$ **then**
    return SQL_{current} ▷ *Sucess*
  end if
  $P_{\text{corr}} \leftarrow \text{Prompt}(S_C, Q, SQL_{\text{current}}, error, dialect)$
  $SQL_{\text{current}} \leftarrow \text{LLM}(P_{\text{corr}})$
  $retry \leftarrow retry + 1$
end while
return None ▷ *Failure*

3. RESULTS AND DISCUSSION

This section details the empirical results of our proposed two-phase workflow. We aim to assess its effectiveness in enhancing the Text-to-SQL capabilities of open-source SLMs and compare the performance of these models with different specializations (general-purpose vs. code-focused) when augmented by our framework.

3.1. Results

To isolate the effect of the proposed workflow, we benchmark it against a minimal *baseline pipeline* that invokes the language model with a single zero-shot prompt. Whenever the authors of a model provide an official prompt as is the case for *Qwen2.5-coder-14B*, we adopt it unchanged; otherwise we fall back to the generic "SQL – Natural Language Translation" template from OpenAI (OpenAI, 2025). Because this baseline neither filters nor enriches the schema and performs no execution-guided repair, it serves as a rigorous reference point.

Model quality is assessed on the Spider *Dev* and *Test* splits with the two canonical Text-to-SQL metrics:

Execution accuracy (EX) - reports the fraction of generated queries that, when executed, return the same result set as the ground truth; any runtime error counts as a failure, so EX reflects functional correctness.

Exact match (EM) is stricter after normalizing trivial differences in case and whitespace, it records the percentage of predictions whose SQL string is identical to the reference, thus capturing syntactic fidelity.

The resulting EX and EM scores for both models under the full two-phase workflow are summarized in Table 1.

Table 1. Workflow Performance on Spider dataset. Baseline results are illustrative estimates (no enrichment and correction). The ↑ indicate improvement over baseline

Method | Models		Spider Dev		Spider Test	
		EX (%)	EM (%)	EX (%)	EM (%)
Baseline	Qwen2.5-coder	81.8	58.8	83.7	57.9
	Phi4	75.1	27.1	77.0	29.2
Workflow	Qwen2.5-coder	**83.1** ↑	**62.3** ↑	**86.2** ↑	**62.2** ↑
	Phi4	76.6 ↑	30.2 ↑	77.9 ↑	31.7 ↑

Table 1. illustrates the impact of the proposed two-phase workflow on two open-source SLMs. For *Qwen2.5-coder*, execution accuracy (EX) climbs from 81.8% to 83.1% on Spider-Dev set and from 83.7% to 86.2% on Spider-Test, while exact-match accuracy (EM) increases from 58.8% to 62.3% and from 57.9% to 62.2%, respectively. The more generalist *Phi-4* model follows the same trend, albeit from a lower baseline, rising from 75.1% to 76.6% EX on Dev and from 77.0% to 77.9% on Test, with EM improving by roughly three percentage points on both splits. In every case the jump in EX outpaces that in EM, confirming that the execution-guided self-correction of Phase II successfully amends syntactic flaws that previously prevented query execution even when the final SQL string diverges from the reference.

Table 2. The comparison of Text-to-SQL performance of our proposed workflow on the Spider dataset with reference methods

Method	Spider Dev		Spider Test	
	EX (%)	EM (%)	EX (%)	EM (%)
(A) Rule-based + Pre-trained Models				
RYANSQL (Choi et al., 2021)	-	58.2	-	66.6
SADGA (Cai et al., 2021)	71.6	-	66.7	29.2
RESDSQL + NATSQL (H. Li et al., 2023)	**84.1**	**80.5**	**79.9**	**72.0**
(B) LLM + In-context Learning				
DIN-SQL (Pourreza & Rafiei, 2023)	-	-	**85.3**	60.0
DAIL-SQL (D. Gao et al., 2023)	86.6	-	84.4	**74.4**
MAC-SQL (Wang et al., 2025)	**86.8**	-	82.8	-
(C) SLM + ICL / Fine-tune				
MSc-SQL (Gorti et al., 2025)	-	-	84.7	-
DTS-SQL (Pourreza & Rafiei, 2024)	**85.5**	**79.1**	84.4	60.0
CodeS (H. Li et al., 2024)	85.4	-	-	-
Proposed workflow	83.1	62.3	**86.2**	**62.2**

A broader comparison in Table 2. positions the workflow among three established families of methods. Against rule-based systems coupled with pre-trained language models, whose best result is 79.9% EX on Spider-Test (RESDSQL + NatSQL), the workflow achieves 86.2%, thereby exceeding every member of that family while relying solely on open-source weights. When set beside GPT-4 systems using in-context learning, it attains a higher Test EX than all published variants—for example, surpassing DIN-SQL at 85.3% and MAC-SQL at 82.8%. Within the SLM category, the strongest prior result is DTS-SQL at 84.7%; the workflow therefore establishes a new state-of-the-art for models of comparable size, demonstrating that carefully engineered prompting plus execution feedback can close, and in this instance invert, the gap to larger proprietary LLMs without costly fine-tuning. The workflow does not yet match the leading EM figures of RESDSQL + NatSQL method (80.5% Dev, 72.0% Test) or DAIL-SQL (74.4% Test), indicating that further work on SQL string canonicalization would be needed to eliminate residual lexical mismatches.

Table 3. Ablation study: Impact of workflow components on EX accuracy (%) of Qwen2.5-Coder:14B model. (SPIDER *Test*)

Method Configuration	Easy	Med	Hard	Extra	All
Full Workflow *(Phase I + II)*	**93.6**	**89.3**	**80.5**	**76.3**	**86.2**
w/o Enrichment *(Phase I)*	93.2	88.6	76.0	69.6	83.8
w/o Self-Correction *(Phase II)*	91.5	84.3	67.2	45.8	77.0
Baseline *(No workflow)*	93.4	87.2	78.6	69.7	83.8

Table 4. Ablation study: Impact of workflow components on EX accuracy (%) of Phi:14B model. (SPIDER *Dev*)

Method Configuration	Easy	Med	Hard	Extra	All
Full Workflow *(Phase I + II)*	87.5	**83.4**	**65.5**	**53.9**	**76.6**
w/o Enrichment *(Phase I)*	89.5	83.0	63.8	47.0	75.5
w/o Self-Correction *(Phase II)*	88.3	81.4	60.3	50.6	74.6
Baseline *(No workflow)*	**90.7**	82.1	58.0	50.9	75.1

The ablation in Table 3. confirms that both phases are indispensable for *Qwen2.5-coder*. Removing schema enrichment leaves overall EX at 83.8% and, more tellingly, drops the *Hard* subset from 80.5% to 76% and the *Extra* subset to 69.6%. Eliminating self-correction is even more damaging: overall EX falls to 77.0%, with *Extra queries* subset collapsing to 45.8 %. The same pattern appears for *Phi-4* in Table 4. The full workflow reaches 76.6% EX, whereas discarding enrichment reduces it to 75.5% and discarding self-correction to 74.6%. Again the steepest degradation is concentrated in the *Extra* tier, where scores drop from 53.9% to 47.0% without enrichment and to 50.6% without self-correction. These observations reinforce the complementary roles of the two phases: enrichment supplies concise, relevant schema context that becomes critical as relational complexity grows, and execution feedback repairs generation errors that SLMs alone cannot consistently avoid.

Taken together, the experiments validate the effectiveness of the two-phase strategy. Across both SLMs the workflow consistently yields higher execution accuracy most notably a 2.5 percentage point gain on *Spider Test* for Qwen2.5-coder and delivers the *highest published Test EX (86.2%)* among systems that eschew closed-source models and extensive fine-tuning. The improvements are most pronounced on the hardest queries, indicating that schema pruning and post-generation self-repair are complementary and jointly indispensable for bringing open-source models to production-grade Text-to-SQL performance.

This section presents the empirical evaluation of our proposed two-phase workflow, designed to enhance the Text-to-SQL capabilities of open-source Small Language Models (SLMs). We analyze the performance improvements achieved by integrating the schema enrichment phase and self-correction mechanisms, utilizing the challenging Spider benchmark.

3.2. Discussion

The results highlight that schema enrichment and execution-driven self-correction are complementary components for improving Text-to-SQL accuracy with open-source small language models. Compared to prior rule-based and LLM-based approaches, the proposed workflow consistently closes the gap with larger proprietary models while maintaining resource efficiency. Notably, the workflow achieves higher execution accuracy than GPT-4–based in-context learning methods on the Spider benchmark, underscoring the potential of carefully engineered prompting combined with lightweight feedback mechanisms.

The ablation analysis further emphasizes that both phases are indispensable. Schema enrichment provides structured and contextually relevant schema knowledge, which becomes increasingly important as query complexity grows, while the self-correction loop effectively mitigates syntactic and semantic errors that would otherwise prevent successful execution. These findings suggest that a hybrid strategy combining structural schema management with dynamic error repair offers a robust pathway for deploying SLMs in production-grade environments.

Limitations: This study's limitations include evaluation primarily on the Spider benchmark, potentially not covering all real-world database complexities. Performance may vary with different SLMs beyond the two tested. The enrichment phase's effectiveness depends on initial metadata

quality and clustering suitability. The self-correction loop has limits in resolving highly complex errors, and computational overhead might be relevant in certain contexts. The evaluation focused on accuracy metrics, omitting latency or deeper query quality analysis.

4. CONCLUSION

This research directly confronts the difficulties in leveraging open-source Small Language Models for practical Text-to-SQL applications, particularly their struggles with complex schemas and context limitations. We introduced and validated a novel two-phase workflow integrating offline, graph-based schema enrichment via Louvain clustering with runtime, execution-driven self-correction. Empirical results on the challenging Spider benchmark demonstrate that this approach markedly improves execution accuracy for diverse SLMs like Qwen2.5-coder and Phi-4. Our work provides not just a conceptual framework but a practical, validated methodology, paving the way for wider deployment of resource-efficient, open-source models in sophisticated natural language database interfaces. Future efforts could focus on extending this methodology to broader database types and further enhancing the robustness and efficiency of the self-correction loop for highly complex queries.

ACKNOWLEDGMENT

This research was supported by The Vietnam National University Ho Chi Minh City - University of Information Technology's Scientific Research Support Fund.

REFERENCES

Banda, F., & Motik, B. (2020). Community-based RDF graph partitioning. *SSWS 2020: Scalable Semantic Web Knowledge Base Systems*, *2757*, 33–48. https://ora.ox.ac.uk/objects/uuid:8835ec45-cf2e-4706-8dac-808f007caa60

Cai, R., Yuan, J., Xu, B., & Hao, Z. (2021). SADGA: Structure-Aware Dual Graph Aggregation Network for Text-to-SQL. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems* (Vol. 34, pp. 7664–7676). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2021/file/3f1656d9668dffcf8119e3ecff873558-Paper.pdf

Cao, Z., Zheng, Y., Fan, Z., Zhang, X., Chen, W., & Bai, X. (2024). *RSL-SQL: Robust Schema Linking in Text-to-SQL Generation* (No. arXiv:2411.00073). arXiv. https://doi.org/10.48550/arXiv.2411.00073

Chen, J., Gan, L., Zhao, Z., Wang, Z., Wang, D., & Zhuang, C. (2025). *SQLCritic: Correcting Text-to-SQL Generation via Clause-wise Critic* (No. arXiv:2503.07996). arXiv. https://doi.org/10.48550/arXiv.2503.07996

Chen, X., Wang, T., Qiu, T., Qin, J., & Yang, M. (2024). *Open-SQL Framework: Enhancing Text-to-SQL on Open-source Large Language Models* (No. arXiv:2405.06674). arXiv. https://doi.org/10.48550/arXiv.2405.06674

Choi, D., Shin, M. C., Kim, E., & Shin, D. R. (2021). RYANSQL: Recursively Applying Sketch-based Slot Fillings for Complex Text-to-SQL in Cross-Domain Databases. *Computational Linguistics*, *47*(2), 309–332. https://doi.org/10.1162/coli_a_00403

Gan, Y., Chen, X., Xie, J., Purver, M., Woodward, J. R., Drake, J., & Zhang, Q. (2021). Natural SQL: Making SQL Easier to Infer from Natural Language Specifications. In M.-F. Moens, X. Huang, L. Specia, & S. W. Yih (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021* (pp. 2030–2042). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.findings-emnlp.174

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., & Zhou, J. (2024). Text-to-SQL empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, *17*(5), 1132–1145. https://doi.org/10.14778/3641204.3641221

Gao, Y., & Luo, Z. (2025). *Automatic database description generation for Text-to-SQL* (No. arXiv:2502.20657). arXiv. https://doi.org/10.48550/arXiv.2502.20657

Gorti, S. K., Gofman, I., Liu, Z., Wu, J., Vouitsis, N., Yu, G., Cresswell, J. C., & Hosseinzadeh, R. (2025). MSc-SQL: Multi-Sample Critiquing Small Language Models For Text-To-SQL Translation. In L. Chiruzzo, A. Ritter, & L. Wang (Eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (pp. 2145–2160). Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.naacl-long.107

Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F., & Huang, X. (2025). *Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL* (No. arXiv:2406.08426). arXiv. https://doi.org/10.48550/arXiv.2406.08426

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., Dang, K., Fan, Y., Zhang, Y., Yang, A., Men, R., Huang, F., Zheng, B., Miao, Y., Quan, S., … Lin, J. (2024). *Qwen2.5-*

Coder Technical Report (No. arXiv:2409.12186). arXiv. https://doi.org/10.48550/arXiv.2409.12186

Li, B., Zhang, Y., Bubeck, S., Pathuri, J., & Menache, I. (2024). *Small Language Models for Application Interactions: A Case Study*. https://doi.org/10.48550/ARXIV.2405.20347

Li, C., Shao, Y., Li, Y., & Liu, Z. (2025). *SEA-SQL: Semantic-Enhanced Text-to-SQL with Adaptive Refinement* (No. arXiv:2408.04919). arXiv. https://doi.org/10.48550/arXiv.2408.04919

Li, H., Zhang, J., Li, C., & Chen, H. (2023). Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(11), 13067–13075. https://ojs.aaai.org/index.php/AAAI/article/view/26535

Li, H., Zhang, J., Liu, H., Fan, J., Zhang, X., Zhu, J., Wei, R., Pan, H., Li, C., & Chen, H. (2024). CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proceedings of the ACM on Management of Data*, *2*(3), 1–28. https://doi.org/10.1145/3654930

Mohammadjafari, A., Maida, A. S., & Gottumukkala, R. (2025). *From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems* (No. arXiv:2410.01066). arXiv. https://doi.org/10.48550/arXiv.2410.01066

Nan, L., Zhao, Y., Zou, W., Ri, N., Tae, J., Zhang, E., Cohan, A., & Radev, D. (2023). Enhancing Text-to-SQL Capabilities of Large Language Models: A Study on Prompt Design Strategies. In H. Bouamor, J. Pino, & K. Bali (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023* (pp. 14935–14956). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-emnlp.996

OpenAI. (2025, June 13). *SQL translation with GPT models*. OpenAI Platform Documentation. https://platform.openai.com/docs/examples/default-sql-translate

Pourreza, M., & Rafiei, D. (2023). *DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction* (No. arXiv:2304.11015). arXiv. https://doi.org/10.48550/arXiv.2304.11015

Pourreza, M., & Rafiei, D. (2024). DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. In Y. Al-Onaizan, M. Bansal, & Y.-N. Chen (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024* (pp. 8212–8220). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.findings-emnlp.481

Qi, J., Tang, J., He, Z., Wan, X., Cheng, Y., Zhou, C., Wang, X., Zhang, Q., & Lin, Z. (2022). RASAT: Integrating Relational Structures into Pretrained Seq2Seq Model for Text-to-SQL. In Y. Goldberg, Z. Kozareva, & Y. Zhang (Eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (pp. 3215–3229). Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.emnlp-main.211

Tai, C.-Y., Chen, Z., Zhang, T., Deng, X., & Sun, H. (2023). Exploring Chain of Thought Style Prompting for Text-to-SQL. In H. Bouamor, J. Pino, & K. Bali (Eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 5376–5393). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.327

Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q.-W., Yin, D., Sun, X., & Li, Z. (2025). *MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL* (No. arXiv:2312.11242). arXiv. https://doi.org/10.48550/arXiv.2312.11242

Wong, A., Pham, L., Lee, Y., Chan, S., Sadaya, R., Khmelevsky, Y., Clement, M., Cheng, F. W. Y., Mahony, J., & Ferri, M. (2024). Translating Natural Language Queries to SQL Using the T5 Model. *2024 IEEE International Systems Conference (SysCon)*, 1–7. https://ieeexplore.ieee.org/abstract/document/10553509/

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., & Radev, D. (2018). Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In E. Riloff, D. Chiang, J. Hockenmaier, & J. Tsujii (Eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 3911–3921). Association for Computational Linguistics. https://doi.org/10.18653/v1/D18-1425