SchemaRAG: A Schema-aware Retrieval-Augmented Generation Framework for Text-to-SQL

Anonymous Author(s)

Abstract

Text-to-SQL refers to the task of converting natural language queries into Structured Query Language (SQL), enabling users to interact with databases without knowing SQL. Large Language Models (LLMs) have demonstrated considerable potential in implementing Text-to-SQL through retrieval-augmented generation and prompt engineering. However, these methods still face challenges in effectively understanding complex database schemas, failing to produce valid SQL queries. To address this issue, this paper proposes a Schema-aware Retrieval-Augmented Generation (SchemaRAG) framework with three core components. First, a SchemaLinker is fine-tuned to align natural language with schema items by knowledge distilling from high-quality chain-of-thought data, where its reasoning capabilities are further refined through group relative policy optimization. Second, a schema-augmented retriever is designed to retrieve the most relevant examples by referencing the database schema, thereby enhancing the LLM's ability to understand and generate SQL syntax. Finally, SchemaRAG adopts a Pareto-optimal selection mechanism to identify the final SQL query from a set of high-quality candidates to enhance robustness. As such, SchemaRAG can effectively learn complex database schemas to syntactically align with the structures of SQL, thereby generating more valid SQL queries. Extensive experiments on two benchmark datasets are conducted across several mainstream LLMs. The results demonstrate that SchemaRAG significantly outperforms four state-of-the-art Text-to-SQL competitors. The source code and datasets of this paper are available at the anonymous link:**https://anonymous.4open.science/r/SchemaRAG-5D6CH66**.

CCS Concepts

• **Information systems** → **Structured Query Language**.

Keywords

Text-to-SQL, Semantic Alignment, Retrieval-Augmented Generation, Large Language Model

ACM Reference Format:

Anonymous Author(s). 2018. SchemaRAG: A Schema-aware Retrieval-Augmented Generation Framework for Text-to-SQL. In *Proceedings of Proceedings of the 2026 ACM SIGMOD/PODS Conference on Management of Data ( )*. ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Text-to-SQL aims to automatically translate natural language questions into executable Structured Query Language (SQL) queries, thereby lowering the technical barrier for non-expert users to perform data operations. Although methods based on Large Language Models (LLMs) have achieved significant progress, they still face a critical challenge: the massive parameter scale of LLMs can introduce content hallucination, which undermines the accurate interpretation of user intent and compromises the precision of SQL generation. To mitigate this issue, retrieval-augmented generation (RAG) and schema linking techniques can be employed to provide grounded context and enhance the model's ability to correctly interpret database structures and user queries.

Query: "How many singers do we have?"
SQL: "SELECT count(*) FROM singer"

Candidate Questions	Semantic Similarity	Corresponding SQL Query
List the name of singers.	0.681	SELECT name FROM singer
What is the number of continents?	0.517	SELECT count(*) FROM continents
How many countries are listed?.	0.608	SELECT count(*) FROM countries

(a) Relying solely on the embedding similarity of questions to select examples may overlook correct examples and instead retrieve incorrect ones.

Query: "How many [ENTITY] do we have?"
SQL: "SELECT count(*) FROM singer;"

Candidate Questions	Semantic Similarity	Corresponding SQL Query
List the name of [ENTITY].	0.705	SELECT name FROM singer;
What is the number of [ENTITY]?	0.882	SELECT count(*) FROM continents;
How many [ENTITY] are listed?	0.929	SELECT COUNT(*) FROM countries;

(b) Applying entity masking to the question can, to some extent, highlight its syntactic structure and help mitigate this issue.

Questions	Semantic Similarity	SQL Query
What is the highest [ENTITY] for [ENTITY] in the [ENTITY] in [ENTITY]?	0.893	SELECT 'Free Meal Count (K-12)' 'Enrollment (K-12)' FROM frpm WHERE 'County Name' = 'Alameda' ORDER BY (CAST('Free Meal Count (K-12)' AS REAL) 'Enrollment (K-12)') DESC LIMIT 1
What is the [ENTITY] of the [ENTITY] that has the highest number of [ENTITY] with an [ENTITY] of over [ENTITY]?		SELECT T2.Phone FROM satscores AS T1 INNER JOIN schools AS T2 ON T1.cds = T2.CDSCode ORDER BY T1.NumGE1500 DESC LIMIT 1

(c) Entity masking in complex queries can cause significant information loss, leading to high similarity between questions with different SQL structures

Figure 1: Limitations of RAG in the Text-to-SQL

RAG provides LLM with a factual basis by retrieving relevant exemplars. However, existing RAG applications in Text-to-SQL suffer from a fundamental limitation. Existing RAG retrieval strategies rely primarily on textual similarity, often neglecting the critical dimensions of database schema and SQL syntactic structure. This text-centric approach often causes the model to retrieve structurally irrelevant examples, which not only fail to offer useful guidance but may also mislead the final SQL generation. Questions can be textually similar yet correspond to vastly different SQL syntactic structures (Figure 1(a)). To mitigate this issue, the technique of entity masking can reduce dependency on specific schema names(Figure 1(b)), but it can also be misleading by erasing key information. As shown in Figure 1(c), two masked questions may appear textually similar, yet one might correspond to a complex aggregate query requiring calculations, while the other is merely a simple filtered retrieval query.

These examples demonstrate that in RAG, relying solely on surface-level textual similarity of questions is insufficient to capture deep structural correspondence of SQL queries effectively. More importantly, as entity masking strips away entity information, it also paradoxically diminishes semantic information carried by the embedding, thereby impairing its ability to represent SQL query intent accurately. Based on these limitations, this study poses its core research question: Can we learn a representation that explicitly encodes the structural properties of SQL queries, thereby enabling the retrieval of syntactically aligned examples that serve as effective templates for the generation model?

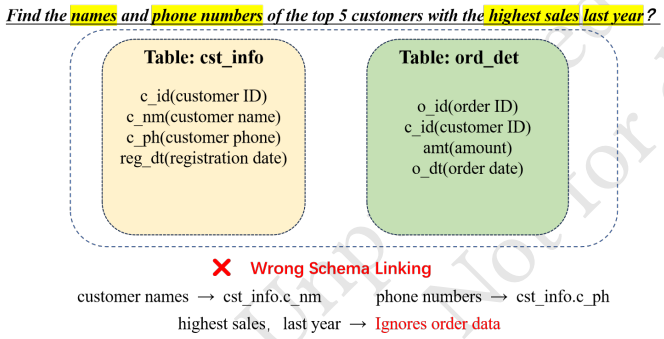


Figure 2: Limitations of schema linking in the Text-to-SQL

Furthermore, schema linking is also a crucial component in addressing the hallucination problem in Text-to-SQL tasks. Existing methods typically approach this issue by framing it as a classification problem, such as using BERT with a classification head [17], or by taking advantage of the few-shot learning capabilities of LLMs[10]. However, these methods struggle with aligning natural language queries to abbreviated or non-standardized schema names. While schema linking techniques based on these approaches can successfully match simple fields like 'customer name,' they fall short when handling more complex concepts, such as 'highest sales' or 'last year,' which require deeper understanding and connection to the orders table (Figure 2).

This research, therefore, investigates whether a joint strategy of Chain-of-Thought (CoT)[32] and Reinforcement Learning (RL)[21] can elevate the schema linking capabilities of LLMs. Can this combined approach achieve a more refined and semantically consistent schema alignment, thereby providing more accurate structural information to support the final SQL generation?

To address the above two key challenges of Text-to-SQL, this paper proposes a Schema-aware Retrieval-Augmented Generation (SchemaRAG) framework. It has three core components. First, SchemaLinker is fine-tuned to map natural language to schema elements by leveraging knowledge distilled from high-quality chain-of-thought examples. Its reasoning capabilities are further enhanced through group relative policy optimization. Second, a Schema-Aware Retriever (SAR) is introduced to select the most relevant examples by utilizing the structure of the database schema, thereby improving the LLM's understanding and generation of SQL syntax. Finally, SchemaRAG designs a Pareto-Optimal SQL Generator (POSG) to choose the best SQL query from a pool of high-quality candidates, increasing overall robustness. As a result, SchemaRAG effectively learns complex database schemas and aligns them syntactically with SQL structures, leading to the generation of more accurate SQL queries. To evaluate the proposed SchemaRAG, we adopt several mainstream LLMs as the base models and compare with four state-of-the-art Text-to-SQL methods. The experimental results on two benchmark datasets show that our SchemaRAG achieves significantly better performance than its peers.

The primary contributions of our work are as follows:

- We propose the SchemaLinker, a module that achieves high-precision schema linking through a rigorously optimized training procedure.
- We propose the SAR, which effectively utilizes database schema structures to learn more accurate, SQL-syntax-aware representations within question embeddings.
- We propose the SchemaRAG framework with its theoretical analyses and algorithm designs.
- Comprehensive experiments are conducted on the benchmark datasets of Spider and BIRD. The results demonstrate that our approach consistently outperforms competitive baselines across multiple evaluation metrics, highlighting its superior generalization capabilities and robustness.

2 Related Works

2.1 Text-to-SQL based on Language Models

Before the rise of LLMs, the predominant approach for text-to-SQL generation involved fine-tuning neural network models with an encoder-decoder architecture. A significant body of research has been dedicated to enhancing the representational capabilities of the encoder. Some studies, for example, encode the question in natural language and the schema of the database by incorporating the structured graph information that exists between query tokens, tables, and columns using Graph Neural Networks to achieve more effective modeling [4, 31]. Other research has focused on injecting SQL grammar into the decoder, which constrains the output space to ensure the syntactic correctness of the generated SQL queries [9, 26, 34]. As language models evolved, researchers began to explore methods for fine-tuning powerful Pre-trained Language Models (PLMs) [16, 25] on input-output sentence pairs [8]. Subsequently, the emergence of competitive LLMs such as GPT [3], Llama [30],

Qwen [2], and DeepSeek [5] has brought revolutionary progress to Natural Language Processing (NLP) tasks, especially those requiring complex reasoning capabilities.

Consequently, a multitude of research directions centered on LLMs have emerged. For example, some studies [22] use models like GPT-4 to decompose the task into a series of simpler subtasks, which are then addressed using a few-shot learning. Other research [7] improves the Text-to-SQL capabilities of LLMs through zero-shot learning, relying on meticulous prompt engineering. Further approaches include utilizing incremental pretraining to increase the SQL generation capacity of LLMs[18], introducing a critic model to select the optimal solution from multiple SQL candidates [12], and integrating the concept of Chain-of-Thought [23, 36] to significantly improve the reasoning and generation capabilities of the models. Additionally, the FinSQL framework [35] presents a model-agnostic approach, using prompt construction, parameter-efficient fine-tuning, and output calibration to efficiently generate accurate SQL queries. This work also belongs to the domain of Text-to-SQL based on language models.

2.2 Retrieval-Augmented Generation

LLMs face challenges such as hallucinations, outdated knowledge, and untraceable reasoning [15, 38]. RAG addresses these issues by retrieving relevant external documents, but retrieval precision and recall limitations can still cause misalignment and information loss.

This RAG-based approach has also demonstrated considerable potential for the Text-to-SQL problem. However, retrieval strategies based on the surface-level similarity of user questions are often inadequate for this task, as textually similar questions can correspond to vastly different SQL structures. To address this, recent works have focused on incorporating SQL structural information. For instance, ACT-SQL[36] employs RAG to construct and select suitable Chain-of-Thought exemplars for the LLM. ReFSQL[37] enhances question representations by using the Tree-Edit-Distance of corresponding SQL queries as a signal for a contrastive learning objective. ASTRES[29] takes a different approach by first generating an approximated SQL query and then re-ranking candidates based on the similarity of their normalized Abstract Syntax Trees (ASTs). While these methods mark a significant step forward, their reliance on surface-level or sequential processing can limit their effectiveness.

Our work is the first to propose a representation learning method, SAR, specifically designed to create novel, Schema-Aware embeddings. By fusing question and schema information through cross-attention and optimizing the resulting embeddings via contrastive learning, SAR directly learns to encode the structural properties of SQL queries for more targeted retrieval of examples.

2.3 Reinforcement Learning Based on Language Models

LLMs offer a promising platform for Reinforcement Learning (RL), which optimizes objectives through interaction-based learning. Foundational methods like DQN [21], PPO [27], and Actor-Critic models [14] laid the groundwork for combining RL with LLMs. To reduce the computational burden of PPO, recent approaches such as Group Relative Policy Optimization have been proposed, as in DeepSeekMath [28].

In the Text-to-SQL domain, the application of RL is constrained by the issue of sparse rewards, particularly when handling complex or deeply nested queries. Although prior research, such as SQL-R1[20], has attempted to mitigate this problem by designing multiple reward mechanisms, models still struggle to learn effective policies from scarce positive feedback, leading to difficulties in training convergence.

To fundamentally address this challenge, our study proposes to decompose the end-to-end SQL generation task, applying RL to the more focused sub-task of schema linking. This decomposition strategy allows the training objective to concentrate more precisely on structural matching and semantic alignment, thereby circumventing the optimization bottleneck caused by sparse rewards in end-to-end training.

3 The Proposed SchemaRAG Framework

3.1 The Overall Structure

The Text-to-SQL task, as shown in Equation 1, translates a natural language question into an executable SQL query based on the database schema (e.g., table names, column names, data types) to accurately capture user intent and retrieve correct results.

$$S = L(Q, D) \tag{1}$$

To reduce hallucination issues in this translation process, our proposed SchemaRAG framework, illustrated in Figure 3, integrates three complementary and mutually reinforcing components designed to enhance SQL query generation.

(1) **SchemaLinker:** The first component is the SchemaLinker, designed to facilitate SQL generation by aligning natural language with schema elements. We fine-tune Qwen-7B in three stages: we first distill CoT data from GPT-4o, then perform multi-task alignment, and finally optimize the model with reinforcement learning.
(2) **SAR:** The second component is a mechanism designed to enhance example selection, taking as input the database schema and the user question. An embedding model first processes these inputs, followed by a cross-attention mechanism to generate Schema-Aware embeddings. Contrastive learning is then applied to both the question embeddings and the Schema-Aware embeddings to improve their representation quality and increase discriminability. Finally, similarity scores are computed based on the enhanced representations to retrieve the top-K most relevant Text-SQL examples.
(3) **POSG:** The third component is a SQL generation and selection module. It first prompts the LLM to generate multiple candidate SQL queries, which are then subjected to multi-dimensional evaluation. Based on these evaluations, the Pareto optimality principle is applied to identify the final, most optimal SQL query.

3.2 SchemaLinker

To enhance the accuracy of the Text-to-SQL task (Equation 1) and mitigate hallucination issues, effective schema linking is essential

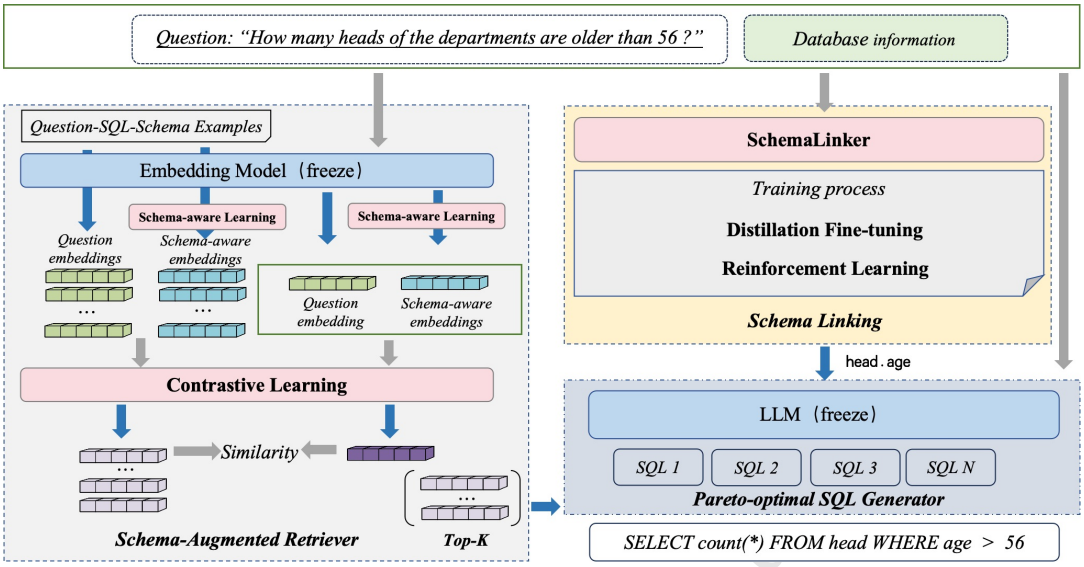


Figure 3: The SchemaRAG architecture is illustrated in the diagram. On the left, the SAR module handles semantic retrieval and contextualized generation. The top-right component represents the SchemaLinker module, which aligns entities in the query with corresponding elements in the database schema. The bottom-right section shows the SQL generation module, responsible for synthesizing the final SQL query based on the outputs of the preceding modules.

for accurately identifying relevant tables and columns from complex database schemas. To address this challenge and ensure precise SQL query generation, we propose a novel database schema representation method, PromptSchema, along with our schema linking model, SchemaLinker.

PromptSchema is a novel database schema representation method designed to improve LLM comprehension of database content. It leverages the BM25S[19] algorithm to automatically select representative samples for each data column, replacing the non-deterministic, LLM-generated descriptions used by methods like M-Schema[10]. PromptSchema guarantees a fully automated and deterministic construction process, providing LLMs with stable, reliable inputs to improve performance on downstream database tasks. The detailed prompts for this method are presented in Appendix H.

A significant challenge in handling complex, real-world databases is to enable LLMs to accurately identify relevant tables and columns from vast database schemas. To address this, a model is needed to predict and select the most critical tables and columns based on a given natural language question. This process guides the model to generate the target SQL query (as referenced in Equation 1) by focusing on the most essential schema information.

To this end, we propose a training method using the Qwen-7B model[33] as the basemodel. This approach aims to enhance the model's ability in natural language understanding and improve its precision in selecting relevant schema elements from the input.

The training process for our SchemaLinker is shown in Figure 4. First, as shown in Equation 2, to ensure the CoT rationale generated by the teacher model (GPT-4o) aligns with the ground truth label, we condition the model on the ground truth label of the given problem.

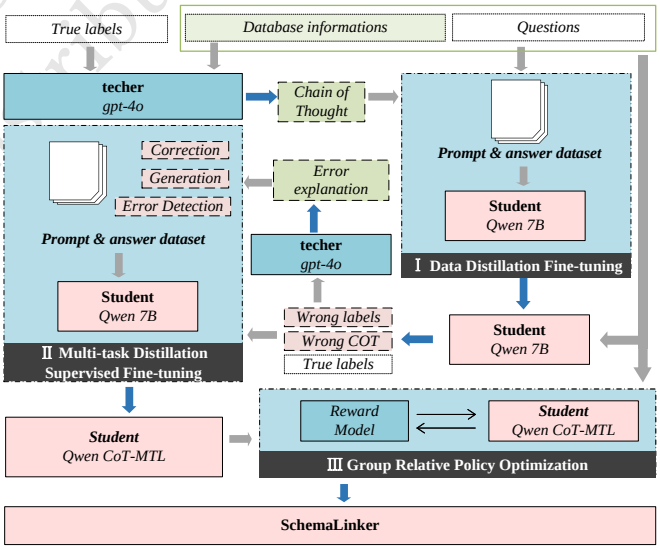


Figure 4: The training process of the schema linking model progresses through three key stages: initial CoT distillation (top right), followed by multi-task distillation (left), and culminating in GRPO-based reinforcement learning (bottom right), ultimately resulting in the SchemaLinker model.

$$\text{For each } (x_i, y_i) \in \{(x_i, y_i)\}_{i=1}^{N} \tag{2}$$

As shown in Equation 3, the CoT and the predicted label are generated by the teacher model.

$$(CoT_i, \hat{y}_i) = L(x_i, y_i) \tag{3}$$

Following the process in Equation 4, we store the resulting CoT and its corresponding predicted label as a single data instance, after confirming their consistency with the ground-truth label.

$$\text{If } \hat{y}_i = y_i \text{ Then } D \leftarrow D \cup \{(x_i, CoT_i, y_i)\} \tag{4}$$

Following the initial training, the fine-tuned Qwen-CoT model was evaluated for five epochs on the rest of the Spider training set. A candidate error set was compiled by taking the union of all samples from each epoch where the predicted labels of the model mismatched the ground-truth labels. However, we observed that in some instances of label mismatch, the model could still generate the correct final SQL query. This necessitated a subsequent manual curation of the candidate set to isolate samples that led to genuinely incorrect SQL outputs, thus forming our refined error dataset. This dataset was then annotated with reasoning paths generated by GPT-4o, which were subsequently corrected and used for a second stage of fine-tuning.

Based on an error-driven learning paradigm, we then designed a multi-task learning framework for the model with three objectives:

(1) **Error Detection:** Identifying and explaining errors within the reasoning process and schema linking.
(2) **Corrective Analysis:** Generating the correct reasoning path and final answer based on the error analysis.
(3) **Answer Generation:** Directly producing the correct reasoning path and answer from the natural language question and database schema.

The goal of this approach is to enhance the primary task proficiency of the model by explicitly training it to recognize and rectify the deficiencies of the model itself.

First, as shown in Equation 5, the input sequence, $\hat{x}_t$, is formed by concatenating a specific task instruction, I_t, with the original input x_t.

$$\hat{x}_t = [I_t \oplus x_t] \tag{5}$$

Then, given the predicted probability distribution of the model, the cross-entropy loss for a single task instance is expressed as:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log P_\theta(y_t | \hat{x}_t, y_{<t}) \tag{6}$$

Moreover, to enable the model to assign task priorities correctly, we control the sampling probability of each task instance using our defined task priority weights (which we set to a ratio of 3:3:10). The probability, p_n, of a single task instance n being sampled in each batch is then given by:

$$p_n = \frac{m_{t_n}}{\sum_{t \in \mathcal{T}} m_t} \cdot \frac{1}{N_t} \tag{7}$$

Consequently, as shown in Equation 8, the expected number of samples for each task i, denoted $E[\text{samples}_i]$, is proportional to the number of instances in the task N_i, and the instance sampling probability p_n.

$$E[\text{samples}_t] \propto N_t \cdot p_n \tag{8}$$

Finally, we set a uniform number of samples for each task. Consequently, the task sampling frequency is controlled solely by our custom-defined weights, enabling the model to focus its learning on high-priority tasks.

Considering that the schema linking task is analogous to mathematical reasoning—where a unique correct answer often exists—we employ the Group Relative Policy Optimization (GRPO) method for the final reinforcement learning fine-tuning of the large model. As shown in Equation 9, for each natural language question and its corresponding database schema, the policy model generates a candidate set of G schema combinations, $\{o_1, o_2, \ldots, o_G\}$, based on the old policy. These candidates are then meticulously evaluated against our defined reward rules to assign specific reward scores. By focusing on the relative performance of the schema links within the group, we guide the policy update according to our established objective:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{\pi_{\text{old}}(o|s)} \Bigg[\frac{1}{G} \sum_{o_i, o_j \in O} \min\left(r_i^{\text{ratio}} A_j,\ \text{clip}(r_i^{\text{ratio}}, 1-\epsilon, 1+\epsilon) A_j\right) - \beta D_{\text{KL}}(\pi_{\text{old}} \| \pi_\theta) \Bigg] \tag{9}$$

To promote structured output, we guide the model to generate responses in a predefined format. The intermediate reasoning steps of the model, derived from prior training. To ensure the transparency of the model's reasoning process and prevent the internalization of computational steps, we enforce a universal format for these reasoning traces. For training stability, as delineated in Equation 10, outputs that do not adhere to the specified format are assigned a zero reward. Consequently, incorrectly formatted outputs preclude the model from receiving any subsequent reward components.

$$S_k = \begin{cases} R, & \text{if format is correct} \\ 0, & \text{if format is incorrect} \end{cases} \tag{10}$$

As shown in Equation 11, the primary reward component uses four weights to control the impact of precision, false positive rate, false negative rate, and F1 score on the final reward. The specific hyperparameter configuration is as follows:

- A **false negative** is assigned the most significant penalty of -3 to strongly discourage the model from omitting any correct schema items and to maximize recall.
- A **true positive** is rewarded with +2 to encourage correct predictions.
- A **false positive** is assigned a minor penalty of -0.5 to maintain a degree of precision without overly constraining the model-inclusive selection strategy.
- An additional weight of +0.5 is applied to the **F1 score** to promote a balanced trade-off between precision and recall.

In essence, this reward structure systematically guides the model to favor inclusion, ensuring that a comprehensive set of schema candidates is selected, which is fundamental to the accuracy of the final query.

$$R = r_c \cdot |P_c \cap T_s| + r_w \cdot |P_s - T_s| + r_m \cdot |T_s - P_s| + w_f \cdot F_1 \tag{11}$$

Finally, the trained SchemaLinker demonstrates excellent performance on the Schema Linking task, effectively and accurately identifying relevant tables and columns for downstream applications. For a comprehensive evaluation of this module—including training details, token consumption, ablation study results, and comparisons across different training stages—please refer to Appendix B.

3.3 Schema-Augmented Retriever

As shown in Figure 5, to enhance the ability of the LLM to generate syntactically correct SQL statements, we adopt the RAG methodology to provide it with few-shot learning examples.

We generated high-quality, Schema-Aware few-shot examples by leveraging GPT-4o in conjunction with our original Question-SQL-Schema pairs. This approach ensures that the retrieved examples are not only semantically relevant to the input question but also syntactically consistent with the target SQL structure.

3.3.1 Schema-Aware Representation Learning . The Schema-Aware embedding learning model is fundamentally based on the attention mechanism. It computes similarity scores between a query and a set of keys to determine the degree of "attention" that should be assigned to the corresponding values. The computation process is formally defined in Equation 12.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{12}$$

First, as shown in Equations 13 and 14, the database consists of a finite set of relations (tables), each of which is characterized by a specific set of attributes (i.e., column names).

$$T = \{t_1, t_2, \ldots, t_n\} \tag{13}$$

$$C = \{c_1^1, \ldots, c_{j_1}^1, c_1^2, \ldots, c_{j_2}^2, \ldots, c_1^n, \ldots, c_{j_n}^n\} \tag{14}$$

To ensure that the embedding of each table is not merely an isolated symbol but is enriched with its internal structure—namely, the columns it contains—we perform an attention operation. As shown in Equation 15, in this step, each table T_i acts as the query and attends to its own set of columns $C_:^i$ which serve simultaneously as the keys and values. This produces a column-aware table embedding T_i^C.

$$T_i^C = \text{Attention}(T_i, C_:^i, C_:^i) \tag{15}$$

As shown in Equation 16, after obtaining table embeddings enriched with column-level information, the next step is to associate them with the specific user question. We perform a second attention operation, where the embedding of the user question serves as the query and attends to all column-aware table embeddings, which act as the keys and values.

$$\hat{S} = \text{Attention}(E_Q, T_:^C, T_:^C) \tag{16}$$

The final embedding is a highly condensed representation that captures the structural information of the database. The goal of training is to ensure that the predicted embedding closely matches the embedding of the ground-truth SQL query in the vector space(Equation 17).

$$\mathcal{L} = \frac{1}{d}\sum_{i=1}^{d}(E_S^i - \hat{S}^i)^2 \tag{17}$$

3.3.2 Contrastive Learning. Upon completion of the training of the Schema-Aware embedding model, we can generate Schema-Aware embeddings based on the input question and schema information. However, these initial schema-aware embeddings still have a key limitation: entity-specific details introduce variance that prevents semantically similar queries from clustering effectively in the embedding space (Figure 8(a)). Consequently, relying solely on these Schema-Aware embeddings for similarity comparison yields suboptimal results.

To address this limitation, we introduce an embedding enhancement model based on contrastive learning. This approach learns a more robust representation by constructing positive and negative sample pairs.

To begin, as shown in Equation 18, we stack the question embedding and the Schema-Aware embedding to form an input sequence of length two.

$$X^{(0)} \in \mathbb{R}^{2\times b\times d} \tag{18}$$

Next, as shown in Equation 19, to regulate the flow of information between the question and schema embeddings, we apply a structured causal mask M, within the self-attention mechanism of the Transformer.

$$M = \begin{pmatrix} 0 & 0 \\ -\infty & 0 \end{pmatrix} \tag{19}$$

This causal mask enforces a unidirectional information flow: the question embedding can attend to the schema embedding, but not vice-versa. This design allows the schema to inform the question's representation without reciprocal distortion.

As shown in Equation 20, the input sequence is then processed through a series of L masked Transformer encoder layers to produce a final output sequence $X^{(L)}$.

$$X^{(l)} = \text{TransformerLayer}(X^{(l-1)}, M) \quad \text{for } l = 1, \ldots, L \tag{20}$$

As shown in Equation 21, from this final output sequence, we extract the first vector, which corresponds to the newly enhanced question representation. A final layer normalization is applied to produce the enhanced embedding E_{final}.

$$E_{\text{final}} = \text{LayerNorm}(X^{(L)}[0, :, :]) \in \mathbb{R}^{b\times d} \tag{21}$$

To train the model and ensure the separability of augmented embeddings from diverse initial pairs, the model incorporates a contrastive learning objective. Furthermore, to promote proximity for augmented embeddings derived from known similar initial pairs, the model also explicitly employs a similarity objective.

Specifically, as shown in Equations 22 and 23, both the anchor sample embedding and the similar sample embedding are fed into the model to generate their respective enhanced embedding representations.

$$E_{\text{main}} = f(E_Q\text{main}, \hat{S}_{\text{main}}) \in \mathbb{R}^{b\times d} \tag{22}$$

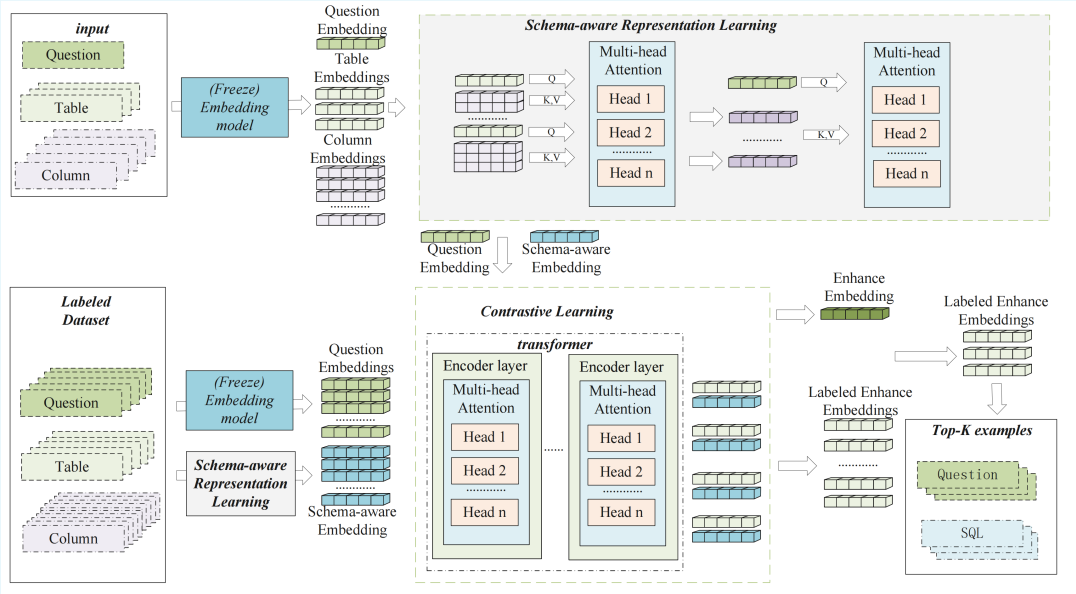


Figure 5: The RAG process involves generating Schema-Aware embeddings via cross-attention and producing enhanced representations with a Transformer, which are then used to retrieve top-k question-SQL examples for few-shot learning.

$$E_{\text{similar}} = f(E_Q \text{sim}_k, \hat{S}_{\text{sim}_k}) \in \mathbb{R}^{b \times d} \quad \text{for } k = 1, \ldots, N_{\text{sim}} \tag{23}$$

To achieve this, as shown in Equation 24, we employ a contrastive loss function that treats other samples within the same batch as negative examples and minimizes their similarity with the anchor sample to enhance the discriminative capability of the model.

$$L_{\text{contrastive}} = -\frac{1}{b}\sum_{i=0}^{b-1}\left(\frac{E_{\text{main},i}\cdot E_{\text{main},i}}{\tau} - \log\sum_{j=0}^{b-1}\exp\left(\frac{E_{\text{main},i}\cdot E_{\text{main},j}}{\tau}\right)\right) \tag{24}$$

In addition, as shown in Equation 25, a similarity loss term is introduced to minimize the distance between the enhanced embedding of the anchor sample and that of its corresponding similar sample, thereby promoting semantic consistency.

$$L_{\text{similarity}} = \frac{1}{N_{\text{sim}}}\sum_{k=1}^{N_{\text{sim}}}\left(1 - \frac{1}{b}\sum_{i=0}^{b-1} E_{\text{main},i}\cdot E_{smiliar,i}\right) \tag{25}$$

Therefore, as shown in Equation 26, the final total loss is computed as a weighted sum of the contrastive loss and the similarity loss, with a fixed similarity loss weight $w_{\text{sim}} = 0.5$. This weighting strategy is designed to balance the trade-off between representation discriminability and semantic alignment.

$$L_{\text{total}} = L_{\text{contrastive}} + w_{\text{sim}} \cdot L_{\text{similarity}} \tag{26}$$

In conclusion, our proposed method integrates a two-stage process to optimize the retrieval of few-shot examples for Text-to-SQL generation. The first stage creates an initial Schema-Aware representation by encoding the database schema's structure relative to the user's question, training it to align with the ground-truth SQL query's embedding. The second stage employs a contrastive learning framework. It takes a combined input of the original question embedding and the Schema-Aware representation, processing them through a masked Transformer to produce a final, enhanced embedding. This contrastive training makes the final representation highly discriminative of the shared SQL syntactic structure between queries, while remaining invariant to superficial, entity-level variations. By leveraging these enhanced embeddings, we accurately retrieve the top-k most similar Question-SQL pairs, providing the LLM with high-quality, contextually-aware examples that guide it to generate SQL queries that are both syntactically correct and precisely aligned with user intent.

3.4 Pareto-optimal SQL Generator

After processing the schema linking and example augmentation tasks, we input the schema links, PromptSchema, and the selected examples into the LLM to generate a set of candidate SQL queries. To further enhance the precision of SQL generation and fully exploit the vast decoding space of LLMs, we perform a multi-dimensional evaluation of the candidate queries. We then apply the Pareto optimality principle to select the most appropriate SQL query based on its executability (S_{ex}), schema linking conformity (S_{sl}), and example consistency (S_{ec}).

$$Examples = \{e_1, e_2, ..., e_n\} \tag{27}$$

In the multi-dimensional evaluation, we first apply a strict filtering step to all generated SQL queries. This step is based on executability (S_{ex}), as defined in Equation 28. Any query that fails

to execute successfully on the database (i.e., $S_{ex}(S) = 0$) will be discarded directly.

$$S_{ex}(S) = \begin{cases} 1 & \text{if } S \text{ is executable without errors} \\ 0 & \text{if } S \text{ is not executable} \end{cases} \tag{28}$$

For the set of SQL queries that pass the executability check, we subsequently evaluate their reliability using the Pareto optimal principle across two dimensions. The first dimension is schema linking conformity (S_{sl}). This score is calculated based on Equation 29 and measures the Jaccard similarity between the schema used in the query and the linked schema.

$$S_{sl}(S, S_k) = \frac{|SchemaUsed(S) \cap S_k|}{|SchemaUsed(S) \cup S_k|} \tag{29}$$

The second dimension is example consistency (S_{ec}). Its calculation is given in Equation 30, which is based on the average tree edit distance of the abstract syntax tree (AST) between the generated query and the example sample set.

$$S_{ec} = \frac{1}{n} \sum_{i=1}^{n} \left(1 - \frac{d(\text{AST}(S), \text{AST}(e_i))}{\max(|\text{AST}(S)|, |\text{AST}(e_i)|)}\right) \tag{30}$$

We then identify the Pareto-optimal set P, which contains all non-dominated queries.A query S_i is considered non-dominated if no other candidate query S_j exists that performs equally or better on all evaluation metrics and strictly better on at least one, as formally defined in Equation 31. This set contains a group of high-quality candidate queries that achieve different "trade-offs" between schema linking conformity and example consistency.

$$\begin{aligned} P = \{S_i \in E \mid \neg\exists S_j \in E \text{ s.t. } (S_{sl}(S_j) \geq S_{sl}(S_i) \wedge S_{ec}(S_j) \\ \geq S_{ec}(S_i)) \wedge (S_{sl}(S_j) > S_{sl}(S_i) \vee S_{ec}(S_j) > S_{ec}(S_i))\} \end{aligned} \tag{31}$$

In summary, this approach enables us to identify a set of candidate queries that, while not superior or inferior, each offer distinct advantages. This ensures that the final selected SQL query is not only executable but also optimal or near-optimal in its core quality dimensions.

3.5 Algorithm Design

According to the above analyses, we designed the *Algorithm SchemaRAG*. Its pseudocode is presented and analyzed in Algorithmic. 1.

Algorithm 1 The overall algorithm workflow of SchemaRAG

Inputs: $Q, DB, Examples$
Output: S
Sample $\leftarrow$ BM25S(Q, DB)
$D \leftarrow$ PromptSchema(Sample, DB)
$S_k \leftarrow$ SchemaLinker(Q, D)
procedure ProcessDataSource(embedding_args...)
    $(T_{out}, C_{out}, Q_{out}) \leftarrow$ Embedding(embedding_args...)
    $T'_{out} \leftarrow$ SchemaAware$(T_{out}, C_{out}, Q_{out})$
    $E \leftarrow$ BuildEnhance(T'_{out}, Q_{out})
    return E
end procedure
$E \leftarrow$ ProcessDataSource(DB, Q)
$E_{example} \leftarrow$ ProcessDataSource(Examples)
$\text{Top}_k \leftarrow$ Similar$(E, E_{example})$
$\{S_1, S^1_{gen}, ..., S^k_{gen}\} \leftarrow$ Generation$(Q, \text{Top}_k, S_k, D)$
$\text{Score}(S^i_{gen}) \leftarrow (S^i_k, S_i, S_c)$
$S_{final} \leftarrow$ SelectFromDominatingSet$(\{S^1_{gen}, ..., S^k_{gen}\})$

4 Experiments

In the subsequent experiments, we aim at answering the following research questions (RQs):

- RQ.1. Does SchemaRAG achieve the best performance among the baseline models considered in this study?
- RQ.2. Does SchemaRAG outperform the models trained through direct fine-tuning?
- RQ.3. What is the impact of self-consistency on the schema linking capability of the SchemaLIinker?
- RQ.4. What factors influence the performance of SAR, and to what extent is the model itself effective?
- RQ.5. Are all the modules in our designed SchemaRAG farmework necessary for the model to function effectively?

4.1 General Settings

4.2 Experiment Setup

4.2.1 Datasets. The core experiments of this study were conducted on two authoritative benchmark datasets, Spider and BIRD, and the performance of our proposed model was evaluated on their respective development sets. Specifically, to train and validate the effectiveness of the RAG module, we constructed two RAG datasets using data generated by GPT-4o and data collected from the internet. A detailed description of these datasets is provided in Appendix C.

4.2.2 Evaluation Metrics. For the Spider dataset, we utilize two primary evaluation metrics: execution accuracy (EX) and execution match accuracy (EM). The EX metric assesses whether the predicted and ground-truth SQL queries produce identical execution results when run against the database. The EM metric for the Spider dataset evaluates if the predicted SQL query is semantically equivalent to the ground-truth query, focusing on structural and semantic equivalence.

The BIRD benchmark also primarily relies on execution accuracy (EX) as its core evaluation metric. This is because the databases in BIRD contain a large volume of values, which minimizes the probability of an incorrect query producing a correct result (i.e., a false positive). In addition to EX, BIRD introduces the valid efficiency score (VES) to assess the execution efficiency of correctly generated SQL queries. However, preliminary experiments indicate that VES is highly susceptible to fluctuations, making EX the most stable and reliable metric for the BIRD benchmark.

4.2.3 Baselines. This study adopts a range of foundation models, including open-source models such as CodeS[18] (*PACMMOD 2024*), XiYanSQL[10] (*Alibaba 2025*), Qwen[33] (*Alibaba 2025*), and Llama [13] (*Meta AI 2024*), as well as powerful proprietary models like GPT-4o-mini and GPT-4o[1] (*OpenAI 2024*), DeepSeek-V3[5] (*DeepSeek 2024*) and GLM-4[11] (*Zhipu AI 2024*). These models are compared against competitive Text-to-SQL methods, including TA-SQL[24] (*ACL 2024*), Din-SQL[22] (*NeurIPS 2023*), ACT-SQL[36] (*EMNLP 2023*), and C3-SQL[7] (*arXiv 2023*). Detailed information about each model can be found in Appendix E.

4.2.4 Implementation Details. Our experiments were conducted on a server equipped with two NVIDIA V100 32GB GPUs, 1024GB of RAM, and an Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz, running Ubuntu 22.04 LTS with PyTorch 2.7.0. We utilized SQLite for all database management. Our embedding model is bge-large-en-v1.5. In our RAG framework, the top-k was set to 3 for retrieval. For the generation phase, we employed beam search to generate five SQL candidates when using open-source LLMs. For closed-source LLMs, we instead adopted a multi-sampling strategy, generating multiple SQL candidates through repeated stochastic decoding. To ensure that the final SQL output achieves an optimal balance across multiple quality dimensions, we adopt the principle of Pareto Optimality, deterministically selecting the final query from a set of high-quality, non-dominated candidate solutions.

4.3 Performance Comparison (RQ. 1)

Table 1: Comparison of model performance on Spider and BIRD datasets

Model	LLM	Spider Dataset		BIRD Dataset	
		EX	EM	EX	VES
TA-SQL	GPT-4o-mini / GPT-4o	85.00	68.70●	52.43	23.21
	Deepseek-v3	84.60	75.80●	56.24	25.66
	Qwen-7B	76.40	19.70	33.11	12.92
	XiYanSQL-14B	76.60	33.60	37.74	17.42
	Llama-8B	72.00	12.60	32.06	11.89
DIN-SQL	GPT-4o-mini / GPT-4o	74.20	60.10	46.86	20.11
	Deepseek-v3	80.60	43.70	47.11	18.81
	Qwen-7B	73.80	27.30	27.50	9.67
	XiYanSQL-14B	85.10	77.00	44.18	22.58
	Llama-8B	73.50	27.90	26.98	10.67
ACT-SQL	GPT-4o-mini / GPT-4o	73.40	45.10	47.51	22.15
	Deepseek-v3	75.50	25.20	45.98	18.80
	Qwen-7B	76.60	23.50	26.27	9.43
	XiYanSQL-14B	75.80	23.70	43.16	20.35
	Llama-8B	76.50	22.80	26.98	9.12
C3-SQL	GPT-4o-mini / GPT-4o	81.90	46.90	44.46	18.86
	Deepseek-v3	76.50	23.50	44.85	18.50
	Qwen-7B	72.30	44.70	28.51	9.67
	XiYanSQL-14B	78.30	48.80	42.34	22.10
	Llama-8B	71.80	44.40	27.08	9.66
SchemaRAG	GPT-4o-mini / GPT-4o	**85.60**	66.80	**68.90**	**31.98**
	Deepseek-v3	**89.60**	75.90	**68.38**	**31.05**
	Qwen-7B	**80.40**	**53.30**	**54.50**	**25.10**
	XiYanSQL-14B	**93.30**	**88.00**	**65.25**	**34.54**
	Llama-8B	**80.80**	**64.50**	**60.37**	**29.92**

Table 2: Loss/Win counts, Wilcoxon signed-rank test, and the Friedman test among models

Model	loss/win	p-value	F-rank
TA-SQL	82/8	1.34×10^{-5}	2.75
DIN-SQL	90/0	1.91×10^{-6}	3.35
ACT-SQL	90/0	1.91×10^{-6}	3.825
C3-SQL	89/1	1.91×10^{-6}	3.975
SchemaRAG	**9/351**	-	**1.1**

Table 1 documents the performance of the investigated text-to-SQL models on the Spider and BIRD datasets, respectively, when integrated with various LLMs. We report the EX and EM on the Spider dataset, alongside the EX and VES on the BIRD dataset. Additional experimental data can be found in Appendix F.

As shown in Table 2, to provide a comprehensive and statistically robust evaluation, we conducted a series of analyses comparing SchemaRAG against the baseline models. This evaluation framework comprises three statistical tests: a Loss/Win analysis, the Wilcoxon signed-rank test, and the Friedman test[6]. The Loss/Win analysis provides a direct frequency count comparing the performance of SchemaRAG with each baseline. For each metric (i.e., EX, EM, and VES), a "win" is recorded when SchemaRAG achieves a higher score than the baseline; otherwise, it is counted as a "loss". To ascertain the statistical significance of these performance differentials, we employed the Wilcoxon signed-rank test, a non-parametric method for pairwise comparison, using the p-value to determine significance. Finally, for a global comparison across all datasets, the Friedman test was utilized to rank all models simultaneously. This test assigns an F-rank value to each model, where a lower rank indicates superior accuracy, thereby providing a holistic assessment of the models' relative capabilities.

SchemaRAG demonstrates substantial improvements over the baseline models considered in this study (Table 1). Statistical analysis further validates the significance of these results (Table 2). In particular, SchemaRAG achieves a dominant win/loss ratio of 351/9 compared to its competitors, with all pairwise comparisons yielding p-values below 0.05 based on the Wilcoxon signed-rank test.

4.4 Comparison with Fine-tuned LLMs (RQ. 2)

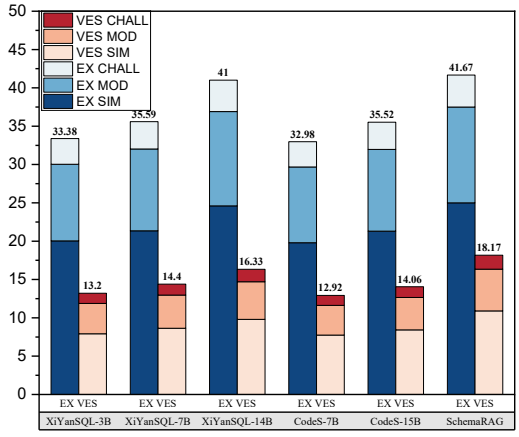


Figure 6: A comparison with fine-tuned large models on the BIRD dataset without evidence.

Figure 6 shows the performance of various models, including the XiYanSQL series, CodeS series, and SchemaRAG(GLM-4-Plus), on the BIRD dataset without evidence, evaluated using both EX and VES metrics.

The figure presents results for each model across three difficulty levels—Simple (SIM), Moderate (MOD), and Challenging (CHALL)—under both EX and VES metrics. Data analysis reveals that **SchemaRAG(GLM-4-Plus)** achieves the highest scores in all

four EX metrics: SIM (50.38), MOD (29.89), CHALL (23.61), and TOTAL (41.66), indicating its superior overall performance. **XiYanSQL-14B** follows closely, demonstrating commendable performance under the EX metric. Similarly, for the VES metric, SchemaRAG(GLM-4-Plus) outperforms other models with leading scores in SIM (22.33), MOD (11.82), CHALL (11.86), and TOTAL (18.16).

In summary, this table compares the performance of our proposed method with various fine-tuned models on the "BIRD(w/o knowledge)" dataset. The comparative analysis of EX and VES metrics unequivocally demonstrates that **SchemaRAG(GLM-4-Plus)** exhibits the most outstanding performance, both in overall efficacy and across specific difficulty levels, significantly surpassing the XiYanSQL and CodeS series models.

4.5 Impact of Self-Consistency on Schema Linking in SchemaLinker (RQ. 3)

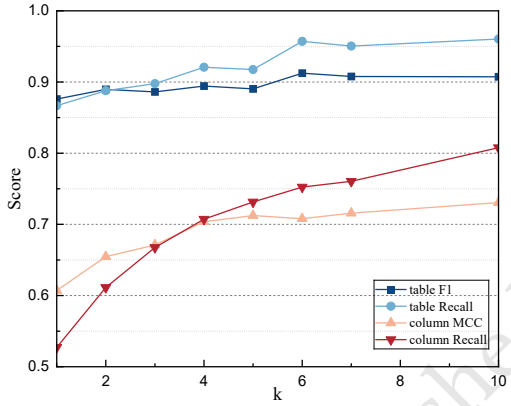


Figure 7: Performance metrics (Table F1, Table Recall, Column MCC, and Column Recall) as a function of the number of self-consistency runs (k).

As shown in Figure 7, to investigate the effect of self-consistency on the SchemaLinker model, we conducted experiments on a challenging subset of 300 samples extracted from the BIRD dataset. For each question, the SchemaLinker model was run k times. The final score was calculated based on the union of all results generated across these k iterations.

Our observations are as follows:

- As the value of k (i.e., the number of self-consistency trials) increases, all four evaluation metrics exhibit a clear upward trend. This indicates that leveraging a self-consistency strategy, by aggregating the union of multiple generated outputs, effectively enhances the performance of the SchemaLinker model.
- The performance improvement is particularly significant as k increases from 1 to approximately 6 or 7. When k is further increased (e.g., beyond 7), the growth rate of the metric scores markedly decreases, and the curve begins to plateau. This suggests that in practical applications, excessively increasing the value of k may yield limited marginal gains, necessitating a careful balance between efficacy and computational cost.

Therefore, the experimental results confirm the effectiveness of the self-consistency method for improving the performance of the SchemaLinker model in extraction tasks, especially when processing high-difficulty samples.

4.6 Factors Influencing the Performance and Effectiveness of SAR (RQ. 4)

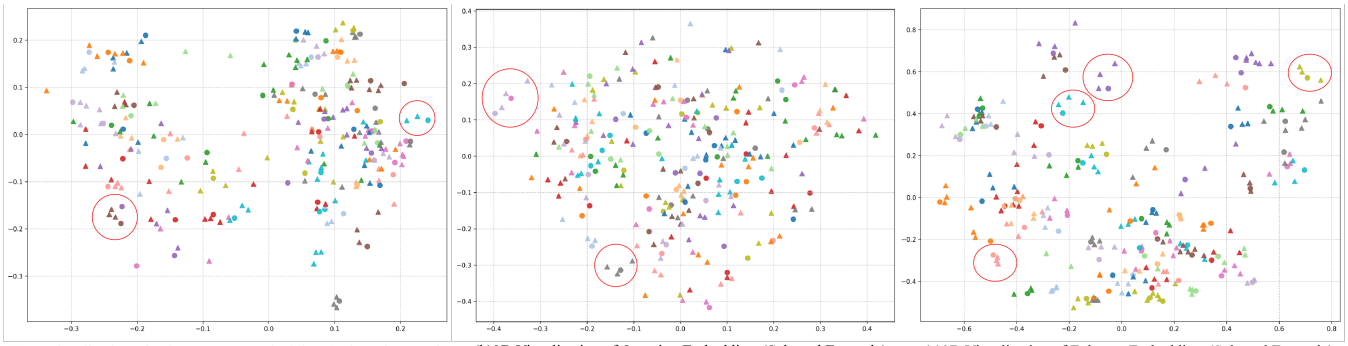
4.6.1 Analysis of Embedding Visualization. As shown in Figure 8(c), the final embeddings, refined by our contrastive learning framework, demonstrate a significantly improved feature space. This enhancement is achieved through a two-stage process. Initially, the schema-aware representation learning captures the essential structural information of the database schema relative to the user's question. Subsequently, the contrastive learning stage operates on these initial representations, dramatically enhancing their discriminability. The result is the effective clustering of samples with similar SQL syntactic structures and the clear separation of dissimilar ones. This well-structured feature space validates our method's efficacy and provides a robust foundation for the subsequent retrieval of high-quality few-shot examples.

4.6.2 Hyperparameter Sensitivity Analysis. To determine the optimal hyperparameters for training, we first conducted a sensitivity analysis on the learning rate and the contrastive temperature. For this analysis, we employed a baseline architecture with 3 Transformer layers and 4 attention heads. The model's performance was primarily evaluated by the F1-score on the positive example selection task.

As shown in Figure 9, we began by tuning the learning rate while keeping the temperature at a default value of 0.05. A learning rate of 10^{-4} demonstrated the best performance on both the RAG_Spider and RAG_BIRD datasets, achieving F1-scores of 0.8554 and 0.8905, respectively. Subsequently, with the learning rate fixed at its optimal value of 10^{-4}, we analyzed the effect of the contrastive temperature. The results indicate that a lower temperature is generally beneficial. Specifically, a temperature of 0.05 yielded the highest F1-score of 0.9006 on RAG_BIRD. For RAG_Spider, while multiple lower temperatures performed similarly, 0.05 also provided a strong F1-score of 0.8554. This suggests that a sharper contrastive objective helps the model learn more discriminative features. Based on these findings, we selected $LR = 10^{-4}$ and $T = 0.05$ for all subsequent experiments.

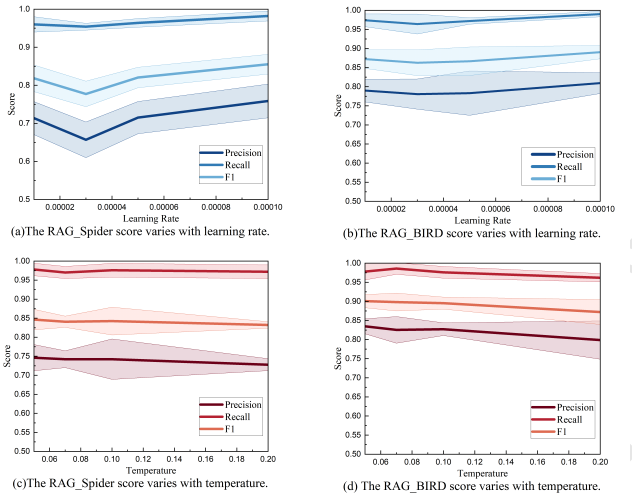
4.6.3 Impact of Model Architecture. we then investigated the impact of the model's architecture by varying the number of Transformer layers and attention heads.

The results, presented in Figure 10, reveal distinct optimal architectures for each dataset. For the more challenging RAG_BIRD dataset, which features greater divergence between questions and schemas, a leaner architecture with 2 Transformer layers and 2 attention heads achieved the best performance. Conversely, for the RAG_Spider dataset, a deeper model with 3 Transformer layers and 8 attention heads was more effective at capturing finer-grained distinctions. This highlights the importance of tailoring model capacity to dataset complexity. Furthermore, we observed that excessively deep or wide architectures led to performance degradation on both datasets, likely due to over-parameterization.



(a)2D Visualization of Schema aware Embeddings(Selected Example) (b)2D Visualization of Question Embeddings(Selected Example) (c)2D Visualization of Enhance Embeddings(Selected Example)

Figure 8: Visualization of different embedding spaces via Principal Component Analysis (PCA). Points of the same color represent pairs with similar SQL syntactic structures. Shapes distinguish between input queries (circles) and candidate examples from our RAG datasets (triangles).



(a)The RAG_Spider score varies with learning rate. (b)The RAG_BIRD score varies with learning rate.

(c)The RAG_Spider score varies with temperature. (d) The RAG_BIRD score varies with temperature.

Figure 9: The performance of SAR with different temperature parameters and learning rates on the RAG_Spider and RAG_BIRD datasets, averaged over 5 runs with mean and variance reported.

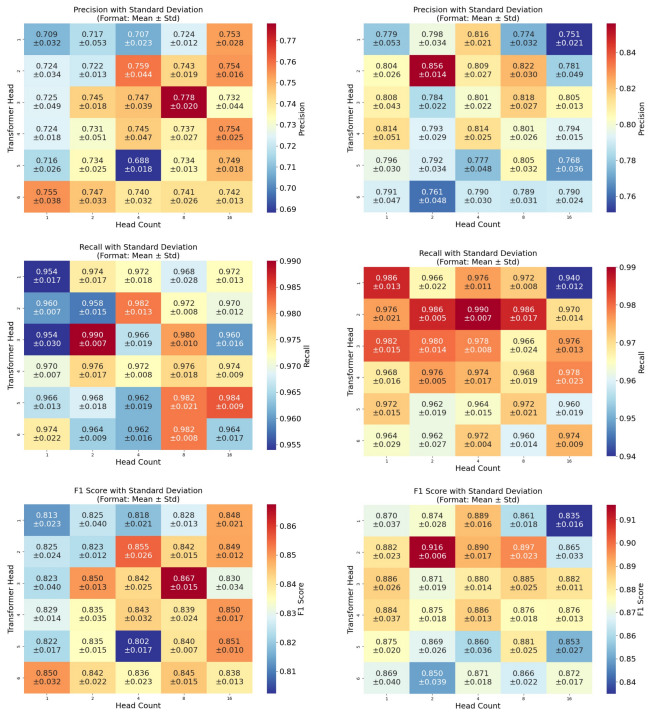


Figure 10: The performance of SAR with different numbers of attention heads and Transformer layers on the RAG_Spider(left) and RAG_BIRD(right) datasets, averaged over 5 runs with mean and variance reported.

4.7 Ablation Experiments (RQ. 5)

To isolate and understand the contribution of each module within our framework, we conducted a comprehensive set of ablation experiments on the Spider and BIRD datasets. We systematically removed three key components: the SchemaLinker, the SAR, and the POSG. The results, detailed in Tables 3 and 4, confirm that the competitive performance of SchemaRAG arises from the powerful synergy between its components. Additional experimental data can be found in Appendix G.

The ablation study indicates that SAR is the most critical component of our framework. As shown in Table 3, removing it causes the largest performance degradation across all models and metrics. For instance, the execution accuracy of GPT-4o on the BIRD dataset drops from 68.90% to 52.54%, and that of Llama-8B drops from 60.37% to 46.41%. On Spider, the EM score of Deepseek-v3 plunges from 75.4% to 20.2%, demonstrating that SAR is essential for generating syntactically correct SQL.

Removing the SchemaLinker module leads to a consistent, though smaller, drop in performance across models and datasets, confirming the importance of SchemaLinker in schema selection. For example, the EX score of XiYanSQL-14B on the BIRD dataset drops

Table 3: Ablation study of SchemaRAG on the Spider and BIRD datasets

Model	LLM	Spider Dataset		BIRD Dataset	
		EX	EM	EX	VES
w/o SchemaLinker	GPT-4o-mini / GPT-4o	85.60	66.90•	65.19	30.80
	Deepseek-v3	85.30	78.10•	64.73	30.93
	Qwen-7B	78.10	53.10	55.22	28.22•
	XiYanSQL-14B	92.10	89.70•	63.30	31.32
	Llama-8B	77.70	60.50	55.35	28.95
w/o SAR	GPT-4o-mini / GPT-4o	79.40	26.70	52.54	23.89
	Deepseek-v3	80.80	20.20	52.09	22.00
	Qwen-7B	72.50	22.00	46.68	21.19
	XiYanSQL-14B	92.60	88.50	61.99	28.88
	Llama-8B	58.90	14.30	46.41	20.12
w/o POSG	GPT-4o-mini / GPT-4o	86.45	65.02	66.80	30.88
	Deepseek-v3	86.60	74.20	66.28	31.00
	Qwen-7B	79.50	53.20	54.12	26.23
	XiYanSQL-14B	92.60	86.50	65.04	33.63
	Llama-8B	79.80	62.24	59.26	29.82
SchemaRAG	GPT-4o-mini / GPT-4o	**87.70**	66.10	**68.90**	**31.98**
	Deepseek-v3	**89.60**	75.40	**68.38**	**31.05**
	Qwen-7B	**80.50**	**53.30**	**55.35**	25.42
	XiYanSQL-14B	**93.30**	88.00	**65.25**	**34.54**
	Llama-8B	**80.90**	**64.50**	**60.37**	**29.92**

Table 4: Loss/Win counts, Wilcoxon signed-rank test, and the Friedman test among models

Model	loss/win	p-value	F-rank
w/o SchemaLinker	59/31	9.40×10^{-3}	1.85
w/o SAR	89/1	3.81×10^{-6}	2.90
w/o POSG	88/2	9.40×10^{-3}	1.66
SchemaRAG	**34/236**	-	**1.25**

from 65.25% to 63.30% without SchemaLinker. Interestingly, minor EM gains are observed on Spider, which may result from the model having access to a broader schema context. In simpler queries, this could occasionally allow the model to match surface forms better, even if the semantic precision suffers. However, such gains are inconsistent and do not outweigh the significant drops in EX, especially on complex datasets, such as BIRD.

The POSG module, positioned as the final stage in the generation pipeline, primarily serves to select the optimal SQL query from multiple candidates. It functions more as a robustness enhancement mechanism for the final output rather than as a core performance-driving engine. This observation is clearly supported by our experimental results: among the three key components, removing the POSG module leads to the smallest drop in performance.

Our analysis demonstrates that the complete SchemaRAG framework achieves competitive performance through the critical and complementary functions of its two core modules: SchemaLinker and SAR. These results are statistically significant and underscore that the success of the framework is fundamentally guaranteed by this effective integration, which addresses the distinct challenges of "what to use" (SchemaLinker) and "how to use it" (SAR) for generating complex and accurate SQL.

The POSG module acts as a final refinement, selecting the most reliable SQL query from the high-quality candidates produced by the core modules. While not central to performance, it further enhances the overall robustness of the framework.

5 Limitations and Future Work

While SchemaRAG has demonstrated competitive performance on the Spider and BIRD benchmarks, we acknowledge the inherent limitations of these datasets and recognize that the framework may face challenges in more complex, real-world scenarios. Future work will be primarily concentrated on the following dimensions:

First, although our proposed SchemaLinker demonstrates commendable performance on the tested benchmarks, scaling it to schemas with thousands of elements presents a clear challenge and an important direction for future work. To ensure robustness, we therefore provide the LLM with both the linked schema items and the complete database schema during the final SQL generation stage.

Second, despite the challenging nature of the Spider and BIRD datasets, they are circumscribed in terms of the scale of their database schemas and the structural complexity of their queries. The core strength of SchemaRAG lies in its ability to retrieve SQL exemplars that are structurally analogous to the input question, thereby providing as effective syntactic templates. However, as the database schema expands to encompass thousands of tables and columns, even Schema-Aware embeddings may struggle to maintain discriminability within a high-dimensional, dense embedding space, leading to a leading to degraded retrieval efficiency and accuracy. More critically, when test queries exhibit entirely novel structures not present in the training corpus (i.e., in a true zero-shot setting for query structures), the framework may fail to identify structurally similar counterparts.

Therefore, a crucial next step involves conducting rigorous empirical evaluations of SchemaRAG on more challenging benchmarks to quantify its current limitations. Concurrently, we will explore the aforementioned research directions to develop a more robust and scalable solution for a broader spectrum of complex, real-world Text-to-Structure tasks.

6 Conclusion

This paper proposes SchemaRAG, an innovative framework designed to address the challenge LLMs face in effectively understanding complex database schemas. Its efficacy stems from the synergy of three core components: the SchemaLinker, which accurately identifies relevant schema elements; the SAR, which retrieves syntactically similar examples to guide the generation process; and a Pareto-optimal selection mechanism, which ensures the robustness of the final output. Experimental results demonstrate the framework's superior performance. The "schema-aware" principle embodied in SchemaRAG not only offers a potent solution to this critical task but also provides valuable insights for broader Text-to-Structure applications.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, et al. 2024. GPT-4 Technical Report. arXiv:2303.08774

[2] Jinze Bai, Shuai Bai, Yunfei Chu, et al. 2023. Qwen Technical Report. arXiv:2309.16609

[3] Tom Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language Models are Few-Shot Learners. In *NeurIPS*, Vol. 33. 1877–1901.

[4] Ruisheng Cao, Lu Chen, Zhi Chen, et al. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In *Proceedings of ACL-IJCNLP*. 2541–2555.

[5] DeepSeek-AI, Aixin Liu, Bei Feng, et al. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437

[6] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *JMLR* 7, Jan (2006), 1–30.

[7] Xuemei Dong, Chao Zhang, Yuhang Ge, et al. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306

[8] Longxu Dou, Yan Gao, Mingyang Pan, et al. 2022. UniSAr: A Unified Structure-Aware Autoregressive Language Model for Text-to-SQL. arXiv:2203.07781

[9] Han Fu, Chang Liu, Bin Wu, et al. 2023. CatSQL: Towards Real World Natural Language to SQL Applications. *Proceedings of the VLDB Endowment* 16, 6 (Feb. 2023), 1534–1547.

[10] Yingqi Gao, Yifu Liu, Xiaoxia Li, et al. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. arXiv:2411.08599

[11] Team GLM, Aohan Zeng, Bin Xu, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. arXiv:2406.12793

[12] Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, et al. 2025. MSc-SQL: Multi-Sample Critiquing Small Language Models For Text-To-SQL Translation. In *Proceedings of NAACL(Long Papers)*, Vol. 1. 2145–2160.

[13] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of ICML*, Vol. 80. 1856–1865.

[15] Nikhil Kandpal, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2023. Backdoor Attacks for In-Context Learning with Language Models. arXiv:2307.14692

[16] Mike Lewis, Yinhan Liu, Naman Goyal, et al. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of ACL*. 7871–7880.

[17] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *Proceedings of the AAAI*, Vol. 37. 13067–13075.

[18] Haoyang Li, Jing Zhang, Hanbing Liu, et al. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 127.

[19] Xing Han Lù. 2024. BM25S: Orders of magnitude faster lexical search via eager sparse scoring. arXiv:2407.03618

[20] Peixian Ma, Xialie Zhuang, Chengjin Xu, et al. 2025. SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning. arXiv:2504.08600

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[22] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *NeurIPS*, Vol. 36.

[23] Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. In *Findings of EMNLP*. 8212–8220.

[24] Ge Qu, Jinyang Li, Bowen Li, et al. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In *Findings of ACL*. 5456–5471.

[25] Colin Raffel, Noam Shazeer, Adam Roberts, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR* 21, 140 (2020), 1–67.

[26] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of EMNLP*. 9895–9901.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347

[28] Zhihong Shao, Peiyi Wang, Qihao Zhu, et al. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv:2402.03300

[29] Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, et al. 2024. Improving Retrieval-augmented Text-to-SQL with AST-based Ranking and Schema Pruning. In *Proceedings of EMNLP*. 7865–7879.

[30] Hugo Touvron, Louis Martin, Kevin Stone, et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288

[31] Bailin Wang, Richard Shin, Xiaodong Liu, et al. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of ACL*. 7567–7578.

[32] Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of NeurIPS*. Article 1800, 14 pages.

[33] An Yang, Baosong Yang, Beichen Zhang, et al. 2025. Qwen2.5 Technical Report. arXiv:2412.15115

[34] Pengcheng Yin and Graham Neubig. 2017. A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of ACL(Long Papers)*. 440–450.

[35] Chao Zhang, Yuren Mao, Yijiang Fan, et al. 2024. FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis. In *SIGMOD*. 93–105.

[36] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. In *Findings of EMNLP*. 3501–3532.

[37] Kun Zhang, Xiexiong Lin, Yuanzhuo Wang, et al. 2023. ReFSQL: A Retrieval-Augmentation Framework for Text-to-SQL Generation. In *Findings of EMNLP*. 664–673.

[38] Yue Zhang, Yafu Li, Leyang Cui, et al. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. arXiv:2309.01219