

SchemaToken: Tokenizing Database Schemas for Efficient and Robust Text-to-SQL Linking

Weibin Liao^{1,2,3}, Jingwen Liu^{1,2,3}, Rihong Qiu^{1,2,3}, Xin Gao^{1,2,3}, Xinyu Ma^{1,2,3},
Hongxin Ding^{1,2,3}, Yue Fang^{1,2,3}, Yifan Zhu⁵, Yuheng Huang⁶,
Junfeng Zhao^{2,3}, Yasha Wang^{1,3,4,✉}

¹National Engineering Research Center of Software Engineering, Peking University

²School of Computer Science, Peking University

³Key Laboratory of High Confidence Software Technologies, Ministry of Education

⁴Peking University Information Technology Institute (Tianjin Binhai)

⁵School of Computer Sciences, Beijing University of Posts and Telecommunications

⁶GRGBanking Equipment Co., Ltd.

✉ liaoweibin@stu.pku.edu.cn, wangyasha@pku.edu.cn

Abstract

Schema linking is a crucial step in Text-to-SQL systems, connecting natural language mentions to relevant database tables and columns. Despite recent advances in LLM-based Text-to-SQL models, current approaches face two fundamental limitations: (1) providing detailed column descriptions inflates the input context, often exceeding the model’s effective sequence length, and (2) predicting multi-token column names increases output complexity, making the task fragile to generation errors. To address these challenges, we propose SchemaToken, which represents each database column as a dedicated token in the LLM’s vocabulary. This design compresses column semantics into single embeddings, reducing context length and simplifying prediction. We introduce a two-stage training paradigm: (i) Semantic Alignment, which enables the LLM to understand the newly added schema tokens via structured embeddings, and (ii) Pseudo-task Alignment, which trains the LLM to generate these tokens as an unordered set using reinforcement learning with a Jaccard-based reward. Experiments on multiple benchmark datasets demonstrate that SchemaToken significantly improves schema linking performance, particularly for large and complex schemas, while maintaining computational efficiency. Our code is publicly available at <https://github.com/MrBlankness/SchemaToken>.

1 Introduction

Schema linking is a fundamental component of Text-to-SQL systems, which enables users to query databases using natural language (Liao et al., 2025b; Gao et al., 2024; Liu et al., 2026). The core objective of schema linking is to identify and connect mentions in a natural language question to relevant tables and columns within a given

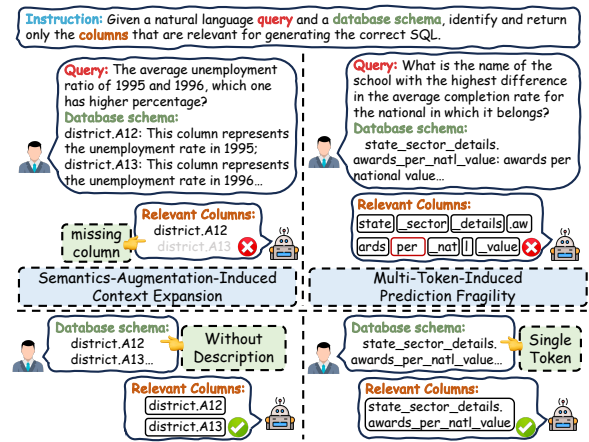


Figure 1: Illustration of key differences between existing schema linking methods (top) and our proposed approach (bottom). **Real-world examples** are taken from the *financial* (left) and *college_completion* (right) databases in the BIRD (Li et al., 2023b) dataset.

database schema (Pourreza and Rafiei, 2023; Qu et al., 2024a; Lee et al., 2025). This task is crucial for accurately translating user queries into executable SQL statements, thereby democratizing data access for non-technical users. The importance of robust schema linking has grown with the increasing demand for intuitive and intelligent database interfaces, particularly in complex enterprise environments where databases often contain hundreds of tables and thousands of columns (Li et al., 2023a).

Recent advancements in Large Language Models (LLMs) (Zhang et al., 2026b; Yang et al., 2026a; Ding et al., 2025) have shown promising results across various natural language processing tasks, including Text-to-SQL generation (Liao et al., 2025b; Gao et al., 2024). For schema linking, LLMs are typically provided with the natural language query, the database schema, and predict the

relevant columns. However, current LLM-based approaches to schema linking face significant challenges.

1. Semantics-Augmentation-Induced Context

Expansion. As illustrated in Fig. 1 (left), schema names in databases often exhibit limited informativeness (e.g., “A1”, “A2”) or rely on domain-specific abbreviations (e.g., “ROI” denoting “Return on Investment” in the *financial* database). Consequently, existing approaches (Wang et al., 2025; Talaei et al., 2024; Pourreza et al., 2025) typically require providing detailed column descriptions to assist LLMs in interpreting the semantics of such schema names. However, this necessity introduces additional context that significantly increases the sequence length, thereby heightening the risk of the lost-in-the-middle (Liu et al., 2024b). Our preliminary experiments reveal a significant decline in schema linking recall as the input column cardinality increases, suggesting limitation in LLMs’ ability to process and effectively utilize very long input sequences for this task.

2. Multi-Token-Induced Prediction Fragility.

When an LLM predicts a column, it must generate the entire column name, token by token. For instance, as shown in the upper-right corner of Fig. 1 (right), predicting a column like “state_sector_details.awards_per_natl_value” requires generating nine consecutive tokens (state, _sector, _details, .aw, ards, _per, _nat, 1, _value) in the correct sequence. This multi-token generation for each column significantly increases the complexity and difficulty of the schema linking task, as a single prediction error (e.g., a missing or incorrect token) renders the entire column prediction incorrect. Our second exploratory experiment indicates that schema linking recall decreases notably as the token length per column increases, further highlighting the output generation limitation.

Our Core Idea: To mitigate these issues, we propose SchemaToken, it represents each database column as a single, dedicated token within the LLM’s vocabulary. By expanding the LLM’s vocabulary to include these unique column tokens, we aim to compress the rich information of each column (its name, description, and semantic context) into a single embedding. This approach directly addresses the input limitation by eliminating the need to pro-

vide explicit column descriptions in the context window, as the LLM would have implicitly learned the column’s meaning through its corresponding token embedding. Simultaneously, it resolves the output limitation: schema linking simply becomes a task of predicting these single column tokens, dramatically simplifying the output space and reducing the chance of generation errors.

To achieve this, we introduce a two-stage training strategy:

1. SchemaToken Understanding via Semantic

Alignment: The first stage focuses on enabling the LLM to understand the meaning encapsulated within each new token. At this stage, we freeze the original LLM and train only the schema-token/codebook embeddings via SFT. However, to overcome the limitations of independent token learning and to capture the relational structure inherent in database schemas (e.g., primary/foreign keys, intra-table column relationships), we draw inspiration from RQ-VAE (Rajput et al., 2023). Instead of a single, independent embedding, each column token’s embedding is composed from a combination of “Table CodeBooks” and “Column CodeBooks” embeddings. This allows us to construct hierarchical and relationally-aware token embeddings, enriching the training information for each token through a more structured data flow.

2. SchemaToken Generating via Pseudo-task

Alignment: The second stage addresses the challenge of enabling the LLM to generate the newly introduced schema tokens in the context of schema linking. To this end, we extend the LLM’s output layer by modifying its language modeling head (LM_Head), such that it explicitly maps hidden representations to the expanded vocabulary, which includes the new schema tokens. During this stage, the learned schema-token/codebook embeddings are frozen, while LoRA/adaptation parameters and the extended LM_Head are updated. Given that schema linking often involves predicting an unordered set of columns, traditional SFT, which forces an ordered generation sequence, is suboptimal. Therefore, we employ Reinforcement Learning (RL), specifically GRPO (Shao et al., 2024; Ding et al., 2026a), for this stage. We design a reward function based on the Jaccard similarity between the predicted set of column tokens and the ground truth set of columns. This RL-based

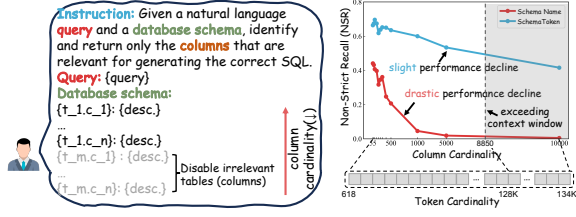


Figure 2: **Semantics-Augmentation-Induced Context Expansion:** Column recall drastically declines as the number of columns increases, highlighting context-length limitations in large-scale databases.

approach allows the LLM to learn to generate the correct set of columns irrespective of their order, directly optimizing for the schema linking objective.

Our Contributions are summarized as follows:

1. **Insightful Analysis:** We provide a detailed analysis of the inherent limitations of current LLM-based schema linking methods, particularly concerning context window length and output generation complexity. Our exploratory experiments empirically validate these limitations, providing strong motivation for our tokenization-based approach.
2. **Methodology Design:** We propose SchemaToken, a novel method to represent each database schema as a unique LLM token. In addition, we designed a two-stage training paradigm, including SchemaToken *Understanding via Semantic Alignment* and SchemaToken *Generating via Pseudo-task Alignment* to assist LLM in understanding and generating these newly added schema tokens in the context of schema linking.
3. **Empirical Demonstration:** We conducted extensive experiments on two publicly available datasets, and the experimental results demonstrated the effectiveness of the SchemaToken and demonstrated its feasibility on large-scale databases,

2 Observation & Motivation

Semantics-Augmentation-Induced Context Expansion. To investigate the relationship between schema linking performance and context length, we conducted an exploratory experiment on the Spider 2.0 (Lei et al., 2025) benchmark. Specifically, we selected *CENSUS_BUREAU_ACS_2* database with 296 tables and 68,434 columns, and simulate

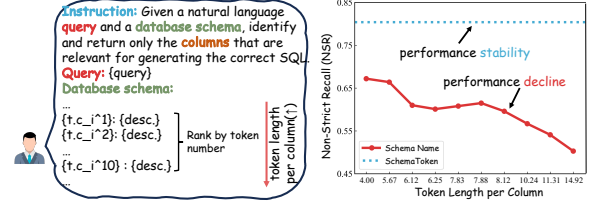


Figure 3: **Multi-Token-Induced Prediction Fragility:** Column recall decreases as per-column token length grows, showing prediction fragility due to multi-token representations.

schema linking scenarios at different column scales by removing tables (& columns) that are irrelevant to the current query. As shown in Fig. 2, the recall of column prediction with Qwen3-8B exhibits a drastic performance decline as the number of columns increases. Moreover, when the number of columns reaches 8,850, the context length already exceeds the 128K window of Qwen3-8B, making schema linking infeasible.

Observation I: *Augmenting schema descriptions inevitably inflates the context length, thereby exacerbating the risk of lost-in-the-middle and significantly degrading schema linking performance, especially in large-scale databases.*

Multi-Token-Induced Prediction Fragility. We further explored the relationship between schema linking performance and the token length of individual columns. To achieve this goal, we employ GPT-4 (Achiam et al., 2023) to generate multiple variants of each column name based on the original column name and its associated description. Using high-temperature sampling, we generate 10 candidate column names per column and then sort these variants by their token lengths, thereby simulating schema linking scenarios under different token budgets. As illustrated in Fig. 3, experimental results demonstrate a consistent performance decline in column recall for Qwen3-8B as the per-column token length grows.

Observation II: *Multi-token representations of columns induce prediction fragility, making schema linking increasingly difficult as the number of required output tokens grows.*

Summary. Taken together, these two exploratory studies highlight fundamental limitations in current schema linking approaches. On the input side, the reliance on column descriptions leads to excessive context expansion, while on the output side, multi-token column representations induce fragile prediction requirements. These observations mo-

tivate us to design a novel solution: **compressing each database column into a single learnable token**. By doing so, we simultaneously mitigate context expansion and prediction fragility, thereby enabling more efficient and robust schema linking with LLMs.

3 Methodology

Table 5 in Appendix B.1 summarizes the key notations used in SchemaToken.

SchemaToken aims to enable large language models (LLMs) to understand and generate database-specific tokens for schema linking tasks, while mitigating the semantic insufficiency and context expansion issues inherent to textual schema descriptions. As illustrated in Fig. 4, SchemaToken consists of three components: (a) framework design with codebooks-based token embedding and extension of LM_head, (b) SchemaToken understanding via semantic alignment, and (c) SchemaToken generating via pseudo-task alignment.

3.1 Task Definition: Schema Linking

Given a natural language query q and a database schema $\mathcal{S} = \{(t_i, \mathcal{C}_i)\}_{i=1}^{\hat{N}}$, where t_i is the i -th table, $\mathcal{C}_i = \{c_{ij}\}_{j=1}^{\hat{M}_i}$ represents its set of columns, $\hat{N}$ is the number of tables in the database and $\hat{M}_i$ is the number of columns in the i -th table. The objective of schema linking is to predict a subset $\mathcal{C}^* \subseteq \bigcup_i \mathcal{C}_i$ that are semantically relevant to q . Formally, the task is defined as:

$$f_\theta : (q, \mathcal{S}) \rightarrow \mathcal{C}^*, \quad (1)$$

where θ denotes the parameters of the LLM. This can be viewed as a multi-label classification or set prediction problem, where the order of predicted columns typically does not matter.

3.2 Framework of SchemaToken

As shown in Fig. 4 (a), SchemaToken expands the LLM vocabulary to include new schema tokens and modifies the language modeling head LM_head to map hidden states into the extended token space, enabling the LLM to both understand and output these schema tokens.

Schema Token Construction. Inspired by Residual Quantized Variational Autoencoders (RQ-VAE) (Rajput et al., 2023), SchemaToken constructs schema tokens through a residual quantization mechanism. Specifically, two L -level

codebooks are introduced: a *Table CodeBooks* $\mathcal{B}_T = \{\mathbf{b}_1^T, \dots, \mathbf{b}_N^T\}$ and a *Column CodeBooks* $\mathcal{B}_C = \{\mathbf{b}_1^C, \dots, \mathbf{b}_N^C\}$, where L is the identifier length and N denotes the codebook size. Each schema token τ is represented by:

$$\mathbf{e}_\tau = \mathbf{e}_{\text{tab}} + \mathbf{e}_{\text{col}}, \quad (2)$$

$$\mathbf{e}_{\text{tab}} = \sum_{l=1}^{L_T} q_T^{(l)}(\mathbf{h}_t), \quad q_T^{(l)}(\mathbf{h}_t) \in \mathcal{B}_T, \quad (3)$$

$$\mathbf{e}_{\text{col}} = \sum_{l=1}^{L_C} q_C^{(l)}(\mathbf{h}_c), \quad q_C^{(l)}(\mathbf{h}_c) \in \mathcal{B}_C. \quad (4)$$

Here, $q_T^{(l)}$ and $q_C^{(l)}$ denote the l -th quantization level functions applied to table and column representations, respectively. This residual quantization mechanism effectively decomposes semantic information into multi-level discrete components, ensuring both compactness and expressiveness. Following the standard setup of RQ-VAE (Rajput et al., 2023), we utilize a random initialization strategy for the codebook embeddings prior to semantic alignment training.

LM_head Extension. To enable the LLM to generate the newly introduced schema tokens, SchemaToken extends the original language modeling head LM_head to map hidden representations into the expanded vocabulary space. Let $\mathcal{V}_{\text{orig}}$ denote the original vocabulary and $\mathcal{V}_{\text{schema}}$ the set of schema tokens, the total vocabulary becomes

$$\mathcal{V}' = \mathcal{V}_{\text{orig}} \cup \mathcal{V}_{\text{schema}}, \quad |\mathcal{V}'| = |\mathcal{V}_{\text{orig}}| + |\mathcal{V}_{\text{schema}}|. \quad (5)$$

The output logits of the LLM are computed as:

$$\mathbf{z} = \text{LM_head}'(\mathbf{h}) = \mathbf{W}'\mathbf{h} + \mathbf{b}', \quad (6)$$

where $\mathbf{h} \in \mathbb{R}^d$ is the hidden state of the final transformer layer, and $\mathbf{W}' \in \mathbb{R}^{|\mathcal{V}'| \times d}$, $\mathbf{b}' \in \mathbb{R}^{|\mathcal{V}'|}$ are learnable parameters of the extended output layer.

To ensure stable adaptation, $\mathbf{W}'$ and $\mathbf{b}'$ are initialized as:

$$\mathbf{W}' = \begin{bmatrix} \mathbf{W}_{\text{orig}} \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{b}' = \begin{bmatrix} \mathbf{b}_{\text{orig}} \\ \mathbf{0} \end{bmatrix}, \quad (7)$$

3.3 SchemaToken Understanding via Semantic Alignment

As illustrated in Fig. 4 (b), SchemaToken aligns the meaning of schema tokens with their corresponding textual semantics via supervised fine-tuning (SFT). During this stage, the original LLM is frozen, and

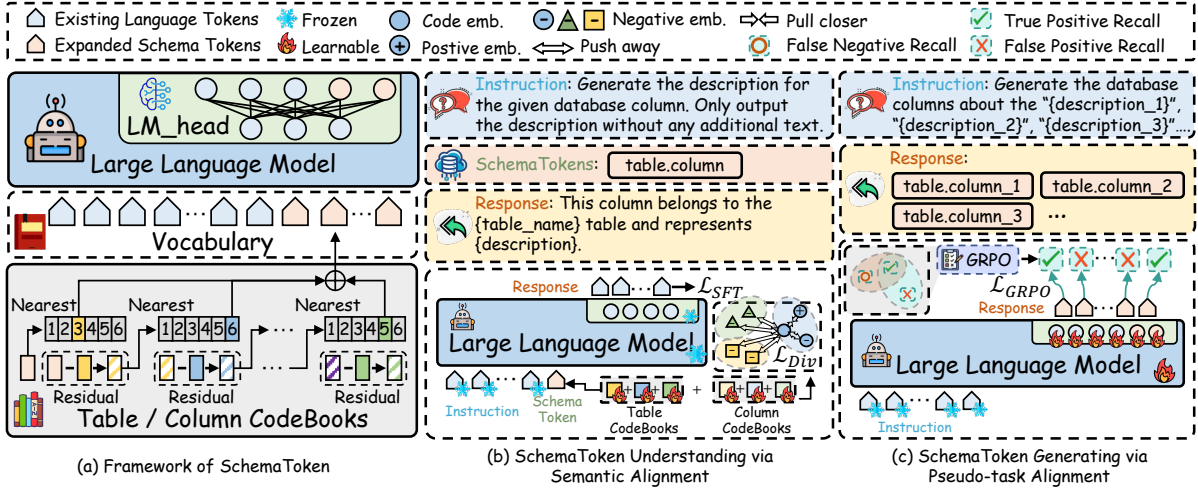


Figure 4: Framework of SchemaToken. (a) illustrates that we extend the LLM’s vocabulary by introducing two codebooks to represent the embedding of each schema token. Meanwhile, we expand the LLM’s LM_head to enable it to generate these newly added schema tokens. (b) illustrates that we leverage the descriptions corresponding to schema tokens to train the model to memorize the semantics of these tokens, while incorporating semantic regularization and diversity regularization to enhance training. (c) illustrates that we employ a pseudo-task analogous to schema linking to train the LLM to generate the extended schema tokens. In this stage, a Jaccard similarity–based reward function is constructed to perform reinforcement learning optimization.

only the schema-token/codebook embeddings in $\mathcal{B}_T$ and $\mathcal{B}_C$ are updated. The learning objective combines two regularization terms: semantic regularization and diversity regularization.

Semantic Regularization. Given a user instruction x that describes a schema token τ , the model is trained to generate the corresponding semantic output y . The SFT loss of semantic is defined as:

$$\mathcal{L}_{\text{Sem}} = -\mathbb{E}_{(x,y)} \sum_{t=1}^{|y|} \log p_{\theta}(y_t | y_{<t}, x), \quad (8)$$

where p_{θ} denotes the LLM output distribution. Multiple prompt templates are used to augment limited schema descriptions, as detailed in Appendix B.3.

Diversity Regularization. To alleviate assignment bias (Wang et al., 2024) within the codebooks, SchemaToken encourages a uniform distribution of embeddings in the latent space. For each codebook $\mathcal{B}$, we perform k -means clustering to obtain K groups $\{\mathcal{G}_1, \dots, \mathcal{G}_K\}$. We then regularize the clustered code embeddings by a diversity loss, which is defined as

$$\mathcal{L}_{\text{Div}} = -\frac{1}{B} \sum_{i=1}^B \frac{\exp(\langle e_c^i, e_+ \rangle)}{\sum_{j=1}^{N-1} \exp(\langle e_c^i, e_j \rangle)}, \quad (9)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product, and B is the batch size, e_c^i is the nearest code embedding

of schema token τ , e_+ denotes the code embedding of a randomly selected sample from the same cluster of code c , and $e_{j \in \{1, \dots, N\} \setminus c}$ represents all code embeddings from the codebook except for e_c . Intuitively, we pull the code embeddings from the same cluster closer and push the embeddings of codes from different clusters away from each other, thus encouraging diversity of the code embeddings and alleviating the code assignment bias issue.

The total training objective for this stage is:

$$\mathcal{L}_{\text{align}} = \mathcal{L}_{\text{Sem}} + \alpha \mathcal{L}_{\text{Div}}, \quad (10)$$

where α balances semantic accuracy and embedding diversity.

3.4 SchemaToken Generating via Pseudo-task Alignment

The second stage focuses on enabling the LLM to output the relevant schema tokens for the schema linking task. Unlike traditional sequence generation, schema linking requires predicting an *unordered set* of columns. Standard SFT typically optimizes for a fixed output sequence, which is sub-optimal for set prediction. To address this, we frame schema linking as a Reinforcement Learning problem and employ Group Relative Policy Optimization (GRPO) (Shao et al., 2024; Liao et al., 2025a; Ding et al., 2026b).

As shown in Fig. 4 (c), we design a *pseudo-task* that mimics multi-column prediction: let the LLM

Methods	Spider-dev					BIRD-dev					Spider 2.0				
	Schema Linking				T2SQL	Schema Linking				T2SQL	Schema Linking				T2SQL [‡]
	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX
RESDSQL	81.7	87.9	20.1	32.7	76.9	48.6	75.7	28.4	41.3	33.5	8.3	34.1	20.1	25.3	6.7
CodeS	80.4	87.4	20.0	32.6	74.6	58.3	77.9	28.1	41.3	36.8	23.3	55.6	27.4	36.7	16.7
<i>Qwen3-1.7B</i>															
Prompt	38.3	60.4	61.9	<u>61.1</u>	41.1	7.5	37.0	53.1	43.6	10.8	12.5	27.0	16.7	20.6	10.0
MCS-SQL	64.5	79.8	47.1	59.2	58.6	41.5	79.2	33.6	47.2	23.6	20.0	50.9	16.0	24.3	13.3
TA-SQL	51.1	65.3	47.0	54.7	22.2	34.2	65.9	74.7	<u>70.0</u>	20.3	1.7	12.9	16.9	14.6	1.7
RSL-SQL	<u>77.3</u>	<u>85.4</u>	38.2	52.8	<u>64.2</u>	<u>79.2</u>	<u>93.6</u>	31.3	46.9	<u>27.3</u>	<u>52.5</u>	<u>70.6</u>	<u>17.0</u>	<u>27.4</u>	<u>20.0</u>
SchemaToken	80.5	88.7	<u>56.4</u>	69.0	67.9	80.6	95.2	<u>61.4</u>	74.7	31.7	63.3	76.8	34.0	47.1	25.8
Impro (%)	4.1	3.9	-8.9	12.8	5.8	1.7	1.7	-17.8	6.6	16.3	20.6	8.8	100.0	72.0	29.2
<i>Qwen3-8B</i>															
Prompt	47.0	67.6	76.0	71.6	55.4	10.5	45.4	74.0	56.3	18.4	9.2	47.1	59.7	<u>52.7</u>	6.7
MCS-SQL	72.9	83.5	66.1	<u>73.8</u>	74.0	47.9	82.3	66.1	73.3	40.6	35.0	71.7	39.8	51.2	18.3
TA-SQL	63.2	75.6	63.3	68.9	42.6	50.7	77.7	80.7	<u>79.2</u>	36.4	10.0	24.5	26.0	25.2	8.3
RSL-SQL	<u>76.8</u>	<u>84.8</u>	58.1	69.0	<u>77.9</u>	<u>82.1</u>	<u>94.4</u>	47.3	63.0	<u>47.9</u>	<u>70.0</u>	<u>84.4</u>	21.7	34.5	<u>21.7</u>
SchemaToken	82.4	88.6	<u>67.4</u>	76.6	80.2	84.2	96.2	<u>74.1</u>	83.7	51.2	80.0	85.5	<u>44.6</u>	58.6	30.0
Impro (%)	7.3	4.5	-11.3	3.8	2.9	2.5	1.9	-8.2	5.7	7.0	14.3	1.3	-25.3	11.3	38.5
<i>LLaMA3.1-8B</i>															
Prompt	38.2	62.4	74.8	<u>68.0</u>	40.8	6.7	42.1	69.0	52.3	16.0	6.7	33.7	<u>38.8</u>	36.1	6.7
MCS-SQL	69.4	82.3	43.9	57.3	63.2	53.6	84.5	45.2	58.9	36.4	23.3	60.0	34.8	<u>44.1</u>	16.7
TA-SQL	47.5	60.7	28.2	38.5	13.8	30.9	56.9	<u>68.5</u>	<u>62.2</u>	10.3	4.2	10.8	12.2	11.5	4.2
RSL-SQL	<u>78.7</u>	<u>85.6</u>	41.3	55.7	<u>71.1</u>	<u>88.2</u>	<u>96.8</u>	32.3	48.4	<u>43.4</u>	<u>61.7</u>	<u>78.4</u>	22.6	35.1	<u>20.0</u>
SchemaToken	82.7	89.4	<u>58.6</u>	70.8	77.8	90.2	97.1	49.1	65.2	48.6	69.2	81.4	45.5	58.4	26.7
Impro (%)	5.1	4.4	-21.7	4.0	9.4	2.3	0.3	-28.8	4.9	12.1	12.1	3.8	17.3	32.5	33.3

Table 1: Performance comparison of various baseline and SchemaToken across three backbone LLMs on three Text-to-SQL (T2SQL) datasets. All evaluation metrics are defined such that higher values indicate better performance ($\uparrow$). The best results are highlighted in **bold**, while the second-best results are underlined within each backbone group. The *Impro (%)* indicates the relative improvement of SchemaToken over the second-best performance. $\ddagger$ indicates that the SQL generator for this set of tasks is GPT-4.

serve as our policy model, π_θ , which takes as input an instruction describing several schema tokens q' ($m \in [\tilde{\epsilon}, \hat{\epsilon}]$), where $\tilde{\epsilon}$ and $\hat{\epsilon}$ denote the lower and upper bounds, respectively, on the number of columns considered in schema linking. The policy outputs a sequence of tokens, which we interpret as a predicted set of schema tokens $\hat{C}$.

Jaccard-based Reward. The reward function $R(\hat{C}, C_{true})$ is designed to directly optimize the set prediction quality. We use the Jaccard similarity between the predicted set of columns $\hat{C}$ and the ground truth set C_{true} :

$$R(\hat{C}, C_{true}) = J(\hat{C}, C_{true}) = \frac{|\hat{C} \cap C_{true}|}{|\hat{C} \cup C_{true}|}$$

The objective of GRPO is to maximize the expected cumulative reward:

$$\max_{\theta} \mathbb{E}_{\hat{C} \sim \pi_\theta} [J(\hat{C}, C_{true})]$$

During this stage, the learned schema-token/codebook embeddings are frozen to

preserve semantic stability, and we update the LoRA/adaptation parameters together with the extended LM_{head} using the GRPO algorithm. We collect trajectories by having the LLM generate sequences of column tokens, which are then evaluated against the ground truth to compute the Jaccard similarity reward. This allows the LLM to learn a policy that effectively identifies and generates the correct set of column tokens, regardless of their generation order, thereby directly optimizing for the schema linking task.

4 Experiment

Detailed experimental setups, including datasets, evaluation metrics, baselines, and implementation details, are provided in Appendix C.

4.1 Experimental Results

Comparison with Recent Literature. As shown in Table 1, we compare SchemaToken with several state-of-the-art approaches across three Text-to-SQL benchmark datasets. To comprehensively

Methods	Spider-dev					BIRD-dev					Spider 2.0				
	Schema Linking				T2SQL	Schema Linking				T2SQL	Schema Linking				T2SQL [‡]
	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX
<i>Full model</i>															
SchemaToken	82.4	88.6	67.4	76.6	80.2	84.2	96.2	74.1	83.7	51.2	80.0	85.5	44.6	58.6	30.0
<i>Tokenization and system integration controls</i>															
w/o SchemaToken (RSL-SQL)	76.8	84.8	58.1	69.0	77.9	82.1	94.4	47.3	63.0	47.9	70.0	84.4	21.7	34.5	21.7
SchemaToken → SFT	78.6	84.5	61.5	71.2	77.9	82.1	94.4	52.3	67.3	48.5	73.4	74.1	30.5	43.2	24.6
SchemaToken → RL	76.8	85.9	63.5	73.0	77.9	83.5	95.4	60.5	74.0	47.9	70.7	85.6	32.6	47.2	21.7
w/o RSL-SQL	75.9	83.5	60.5	70.2	76.4	80.6	93.4	58.5	71.9	45.8	75.4	85.5	38.8	53.4	26.6
<i>Training objective controls</i>															
RL → SFT	78.9	85.2	60.4	70.7	78.5	82.6	95.1	56.8	71.1	48.5	72.6	85.1	23.8	37.2	24.6
Jaccard → Binary Reward	80.5	84.4	64.2	72.9	78.7	80.6	90.7	70.2	79.1	46.0	67.5	74.2	32.6	45.3	20.0
Jaccard → F1 Reward	81.3	82.0	56.4	66.8	76.4	82.1	91.8	76.2	83.3	50.5	77.5	82.5	45.1	58.3	26.6
<i>Embedding and codebook controls</i>															
w/o Semantic Alignment	67.1	70.2	42.4	52.9	61.0	65.3	71.6	46.9	56.7	31.7	56.7	64.4	42.8	51.4	10.0
w/o Table CodeBooks	81.7	86.2	68.1	76.1	79.8	83.7	94.3	75.0	83.5	50.6	75.0	79.1	44.0	56.5	26.7
w/o Diversity Regularization	81.3	84.5	69.5	76.3	79.7	83.1	92.9	75.9	83.5	49.8	75.8	78.6	45.8	57.9	25.0

Table 2: Ablation and component isolation studies of SchemaToken using *Qwen3-8B* as backbone LLM on three Text-to-SQL (T2SQL) datasets. Rows are grouped by their experimental purpose: removing the complete tokenization paradigm or its system integration, changing the training objective, and ablating embedding/codebook components. All evaluation metrics are defined such that higher values indicate better performance ($\uparrow$). The best results are highlighted in **bold**. $\ddagger$ indicates that the SQL generator for this set of tasks is GPT-4.

evaluate the effectiveness of SchemaToken, we employ backbone LLMs with diverse architectures and scales. Due to the fact that small-parameter models achieve near-zero SQL generation performance on Spider 2.0¹, we adopt a backbone model as the generation model for schema linking and use GPT-4 as the Text-to-SQL generation model for the Spider 2.0 task. In the schema linking task, experimental results demonstrate that SchemaToken outperforms the baselines under the same backbone on the SRR, NSR, and NSF metrics, with the only exception being the NSP metric, where it falls short of some alternatives. This indicates that SchemaToken predicts more columns compared to other methods, resulting in higher recall (as reflected in SRR and NSR) but a lower precision (as reflected in NSP). Nevertheless, the superior performance of SchemaToken on the NSF metric highlights the effective trade-off it achieves in this task. Notably, as schema linking serves as a prerequisite step for Text-to-SQL, recall—particularly SRR—is critical, since missing relevant columns can lead to failures in subsequent Text-to-SQL execution. The results on the Text-to-SQL task further confirm this, with SchemaToken achieving optimal performance across all backbone-controlled comparisons. Overall, compared to the second-best performance within each backbone group, SchemaToken yields an average relative improvement of **7.8% $\uparrow$** in SRR,

3.4% $\uparrow$ in NSR, **17.1% $\uparrow$** in NSF and **17.1% $\uparrow$** in EX across all settings, confirming its strong performance. We further analyze the effect of irrelevant-column noise in Appendix D.4 and Fig. 8, which shows that moderate precision degradation has a much smaller downstream impact than missing required columns. A finer-grained analysis by database scale is provided in Appendix D.3, showing that the advantage of SchemaToken becomes especially pronounced on large schemas.

Ablation Studies. We verify the effectiveness of SchemaToken through the hierarchical ablation and component-isolation study in Table 2, using Qwen3-8B as the backbone LLM across three benchmark datasets. First, the tokenization and system-integration controls isolate where the overall gain comes from. The *w/o* SchemaToken (*RSL-SQL*) row replaces the full paradigm with the strongest backbone-controlled baseline, showing that database-specific schema tokenization improves SRR by **5.6% $\uparrow$** on Spider-dev and **10.0% $\uparrow$** on Spider 2.0. The SchemaToken → *SFT* and SchemaToken → *RL* rows remove schema-specific tokens while retaining the corresponding training strategy and directly predicting original column names; their weaker SRR, NSF, and EX results show that the gains are not merely due to extra SFT/RL training, but are closely tied to the schema-token representation. The *w/o* *RSL-SQL* row removes the integration with RSL-SQL and demonstrates that SchemaToken remains effective as a

¹<https://spider2-sql.github.io/>

Methods	Train Time (Stage 1)	Train Time (Stage 2)	Infer Latency (Input Tokens)	Infer Latency (Output Tokens)	Memory Usage
Prompt	N/A	N/A	18.0k	128.5	26.7G
MCS-SQL	N/A	N/A	1080.5k	7710.3	26.7G
TA-SQL	N/A	N/A	56.1k	136.8	42.8G
RSL-SQL	N/A	N/A	42.3k	532.4	31.9G
SchemaToken	0.1h	4.3h	1.5k	16.7	16.5G

Table 3: Computational overhead of SchemaToken and baseline methods using Qwen3-8B on Spider 2.0.

standalone schema linker while also being complementary to existing strong systems. Second, the training-objective controls compare unordered-set optimization choices. Replacing RL with SFT degrades performance because SFT imposes an arbitrary output order on schema columns, whereas RL directly optimizes the unordered prediction objective. Within RL, the Jaccard reward outperforms both the binary reward and the F1 reward on most recall, F1, and downstream EX metrics; the binary reward is especially sparse because the model receives positive feedback only when all relevant columns are predicted exactly. Third, the embedding and codebook controls validate the internal design of SchemaToken. Removing Semantic Alignment produces the largest performance drop, indicating that the LLM must first understand the meanings of the newly introduced schema tokens. Removing Table CodeBooks also hurts performance because column tokens lose table-level structural information. Removing Diversity Regularization improves precision in some cases but reduces recall-oriented and downstream metrics, suggesting that balanced codebook usage is important for robust schema linking.

Computational Overhead Analysis. We evaluate the computational overhead of SchemaToken against baseline methods in Table 3. While SchemaToken incurs a one-time training cost (4.4 hours in total across two stages), it achieves massive inference efficiency gains by eliminating verbose context. Specifically, compared to the prompt-based baseline, SchemaToken reduces the input tokens from 18.0k to just 1.5k and the output tokens from 128.5 to 16.7. This drastic reduction in sequence length leads to a significant decrease in inference latency and peak memory usage (from 26.7G to 16.5G). Other baselines like MCS-SQL and TA-SQL incur even higher inference overheads. Thus, SchemaToken trades a modest one-time training cost for highly efficient and scalable inference.

Context Expansion & Prediction Fragility of SchemaToken. As shown in Fig. 2 and Fig. 3, we

also analyze SchemaToken in terms of Context Expansion and Prediction Fragility. Fig. 2 indicates that although LLM performance decreases as the number of columns increases, the decline is relatively moderate compared to traditional methods that require adding descriptive information. Furthermore, SchemaToken significantly reduces the context window, enabling the schema linking task to scale to larger databases. Fig. 3 shows that the performance of SchemaToken does not vary with the number of tokens in column names. This is because, regardless of the number of tokens a column name contains, SchemaToken compresses it into a single token, making the prediction complexity independent of token count.

Visualization and Interpretability of SchemaToken Embeddings To understand the "black box" of our learned embeddings, we analyzed the token embeddings post-training. We extracted the high-dimensional embeddings of SchemaToken and applied t-SNE dimensionality reduction to project them onto a 2D plane. Specifically, we conducted our analysis on the formula_1 database from the BIRD dev dataset, which contains 13 tables and 92 columns.

As clearly illustrated in Fig. 5, tokens belonging to the same table naturally cluster together in the latent space. The experimental results demonstrate that SchemaToken successfully aggregates columns from the same table into closely bound spatial neighborhoods. In stark contrast, the ordinary RL strategy (shown on the left) completely fails to produce this structural clustering effect. Furthermore, SchemaToken accurately maps identical columns across tables (e.g., results.constructorId and constructors.constructorId) into adjacent spaces, capturing semantic overlaps like primary/foreign keys. Unlike the naive RL strategy which struggles with consistent semantic mapping, our method effectively captures both structural constraints and semantic similarities simultaneously.

5 Discussion and Conclusion

Global Information in Schema Linking. RAG is generally suitable for retrieving independent, self-contained documents. However, tables and columns within a relational database exhibit a high degree of interdependence. Schema linking requires reasoning over the global database structure. For instance, linking a natural language men-

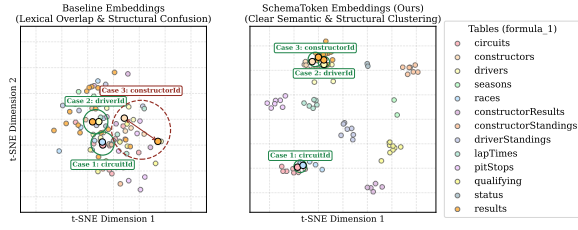


Figure 5: t-SNE visualization of the learned embeddings for the formula_1 database. **Left:** The naive RL strategy fails to cluster columns by tables and struggles with consistent semantic mapping. **Right:** SchemaToken embeddings form distinct, tightly grouped clusters for different tables while preserving relative semantic distances between related domains (e.g., identical columns acting as foreign keys), demonstrating strong structural and semantic discriminative capability.

tion like “student age” might necessitate joining a student table with a demographics table via foreign keys. RAG frameworks compute similarities between the user query and individual columns in strict isolation, completely ignoring the global structural paths (e.g., primary/foreign key relationships) of the database. Such localized similarity matching frequently retrieves columns that are lexically plausible but structurally disconnected, resulting in irrational retrieval decisions and fatal SQL errors. Consequently, empirical evidence (Maa-mari et al., 2024; Zhang et al., 2025) and our experiments (see Appendix D.8) demonstrate the ineffectiveness of such approaches, confirming that RAG is fundamentally unsuited for the schema linking task.

When to Use Prompt vs. SchemaToken. Unlike existing prompt-based methods, SchemaToken requires a two-stage training process tailored to the target database. Therefore, the choice between these paradigms depends on the operational scenario: in ad-hoc query scenarios or rarely accessed personal databases, the zero training overhead of prompt-based methods makes them more favorable; conversely, in production enterprise environments or BI platforms where the system is frequently queried, SchemaToken is vastly superior due to its significantly lower inference latency and memory costs, which quickly amortize the one-time training overhead. This design does not assume that database contents are static: inserting, deleting, or updating records does not affect the learned schema tokens. Adaptation is only needed when the schema itself evolves, and in such cases SchemaToken can update the affected schema to-

kens rather than rebuilding the entire system from scratch, as analyzed in Appendix D.7. We provide a detailed break-even analysis quantifying this trade-off in Appendix F.

Conclusion. In this work, we presented a novel tokenization-based approach, SchemaToken, for schema linking in Text-to-SQL systems. By representing each database column as a single LLM token and employing a two-stage training paradigm, SchemaToken effectively mitigates both context expansion and multi-token prediction challenges inherent in current LLM-based methods. Our experiments show that SchemaToken consistently improves schema linking recall and robustness, particularly in large-scale and complex database scenarios.

Limitations and Future Work

While SchemaToken effectively mitigates input expansion and output fragility in LLM-based schema linking, it introduces a one-time training overhead. Consequently, its zero-shot applicability to entirely unseen databases is constrained compared to training-free prompt-based methods, as applying SchemaToken to a new database currently requires performing the two-stage alignment. This makes SchemaToken most suitable for relatively stable, high-query-volume databases, and less suitable for one-off exploratory settings or databases whose schemas change too frequently. Additionally, the performance of SchemaToken relies to some extent on the quality of column descriptions used during the Semantic Alignment stage, although our experiments demonstrate robustness to variations in description generators. Finally, regarding data dependency, the exact training corpora of many LLMs remain opaque, making absolute verification of data contamination challenging.

Future work will focus on removing the retraining requirement for new databases, potentially by developing a universal schema encoder that can map any unseen schema into the SchemaToken space in a zero-shot manner. We also plan to explore applying this single-token compression concept to other structured reasoning and retrieval tasks, such as API tool selection, Knowledge Graph entity linking, and federated database querying under privacy-preserving constraints.

Ethical Considerations

This work aims to improve the interpretability and efficiency of LLM-based database querying, thereby enhancing data accessibility for non-technical users. However, extending LLM vocabularies to encode private or proprietary database schemas raises potential privacy concerns if such models are not properly isolated or anonymized. We emphasize that all experiments in this paper are conducted on publicly available and open-source datasets and models, ensuring transparency, fairness, and reproducibility. We encourage future applications of SchemaToken to follow responsible data governance practices and comply with institutional and legal standards regarding data confidentiality and user privacy.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No.62576013)

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.
- Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *Association for Computational Linguistics (ACL)*, pages 4560–4565.
- Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:7664–7676.
- Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. Rsl-sql: Robust schema linking in text-to-sql generation. *arXiv preprint arXiv:2411.00073*.
- Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu. 2021. Shadowgnn: Graph projection neural network for text-to-sql parser. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 5567–5577.
- Hongxin Ding, Yue Fang, Runchuan Zhu, Xinke Jiang, Jinyang Zhang, Yongxin Xu, Weibin Liao, Xu Chu, Junfeng Zhao, and Yasha Wang. 2025. 3ds: Medical domain adaptation of llms via decomposed difficulty-based data selection. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 19473–19495.
- Hongxin Ding, Baixiang Huang, Yue Fang, Weibin Liao, Xinke Jiang, Jinyang Zhang, Yinghao Zhu, Zheng Li, Liantao Ma, Junfeng Zhao, and 1 others. 2026a. Promed: Shapley information gain guided reinforcement learning for proactive medical llms. In *Association for Computational Linguistics (ACL)*, pages 32481–32515.
- Hongxin Ding, Baixiang Huang, Yue Fang, Weibin Liao, Zheng Li, Jinyang Zhang, Zhijing Wu, Junfeng Zhao, and Yasha Wang. 2026b. Evorubrics: Dynamic rubrics as rewards via adversarial co-evolution for llm reinforcement learning. *arXiv preprint arXiv:2606.23038*.
- Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *International Conference on Very Large Data Bases (VLDB)*, 17(5):1132–1145.
- Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, and 1 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.
- Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Association for Computational Linguistics (ACL)*, pages 4524–4535.
- Shashank Mohan Jain. 2022. Hugging face. In *Introduction to transformers for NLP: With the hugging face library and models to solve problems*, pages 51–67. Springer.
- Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *International Conference on Computational Linguistics (COLING)*, pages 337–353.
- Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, and 1 others. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *International Conference on Learning Representations (ICLR)*.
- Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. *arXiv preprint arXiv:2502.17248*.
- Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsq: Decoupling schema linking and skeleton parsing for text-to-sql. In *Conference on Artificial Intelligence (AAAI)*, 11, pages 13067–13075.

- Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. [Codes: Towards building open-source language models for text-to-sql](#). *Conference on Management of Data (SIGMOD)*, 2(3):127.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. [Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Weibin Liao, Xu Chu, and Yasha Wang. 2025a. Tpo: Aligning large language models with multi-branch & multi-step preference trees. In *International Conference on Learning Representations (ICLR)*, volume 2025, pages 26698–26720.
- Weibin Liao, Xin Gao, Tianyu Jia, Rihong Qiu, Yifan Zhu, Yang Lin, Xu Chu, Junfeng Zhao, and Yasha Wang. 2025b. Learnat: Learning nl2sql with ast-guided task decomposition for large language models. *arXiv preprint arXiv:2504.02327*.
- Weibin Liao, Jian-guang Lou, and Haoyi Xiong. 2026. Hyfunc: Accelerating llm-based function calls for agentic ai through hybrid-model cascade and dynamic templating. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 819–830.
- Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.
- Jingwen Liu, Weibin Liao, Xin Gao, Junfeng Zhao, and Yasha Wang. 2026. Database context compression for text-to-sql on real-world large databases. *arXiv preprint arXiv:2606.28601*.
- Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024b. Lost in the middle: How language models use long contexts. *Transactions of the Association of Computational Linguistics (TACL)*, 11:157–173.
- Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. In *International Conference on Learning Representations (ICLR)*.
- Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:36339–36348.
- Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024a. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *Findings of Association for Computational Linguistics (ACL)*, pages 5456–5471.
- Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024b. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. *arXiv preprint arXiv:2405.15307*.
- Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, and 1 others. 2023. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:10299–10315.
- Yuxin Ren, Qiya Yang, Yichun Wu, Wei Xu, Yaolong Wang, and Zhiqiang Zhang. 2024. Non-autoregressive generative models for reranking recommendation. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 5625–5634.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and 1 others. 2025. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *International Conference on Computational Linguistics (COLING)*, pages 540–557.
- Tianlong Wang, Yuhang Wang, Weibin Liao, Xin Gao, Xinyu Ma, Yang Lin, Yasha Wang, and Liantao Ma. 2026. Search for truth from reasoning: A dynamic representation editing framework for steering llm trajectories. *arXiv preprint arXiv:2606.28589*.
- Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng

Chua. 2024. Learnable item tokenization for generative recommendation. In *International Conference on Information and Knowledge Management (CIKM)*, pages 2400–2409.

Zhibang Yang, Xinke Jiang, Rihong Qiu, Ruiqing Li, Yihang Zhang, Yue Fang, Yongxin Xu, Hongxin Ding, Xu Chu, Junfeng Zhao, and 1 others. 2026a. Dfams: Dynamic-flow guided federated alignment based multi-prototype search. In *Association for Computational Linguistics (ACL)*, pages 24728–24752.

Zhibang Yang, Xinke Jiang, Yuzhen Xiao, Ruizhe Zhang, Yue Fang, XinFei Wan, Zhengxing Song, Yuxuan Liu, Yuheng Huang, Xu Chu, and 1 others. 2026b. Scaffoldagent: Utility-guided dynamic outline optimization for open-ended deep research. *arXiv preprint arXiv:2606.20122*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. **Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task**. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 3911–3921. Association for Computational Linguistics.

Jinyang Zhang, Hongxin Ding, Yue Fang, Weibin Liao, Muyang Ye, Junfeng Zhao, and Yasha Wang. 2026a. The tell-tale norm: ℓ_2 magnitude as a signal for reasoning dynamics in large language models. *arXiv preprint arXiv:2606.06188*.

Jinyang Zhang, Yue Fang, Hongxin Ding, Weibin Liao, Muyang Ye, Junfeng Zhao, Yasha Wang, and Xu Chu. 2026b. Adept: Continual pretraining via adaptive expansion and dynamic decoupled tuning. In *International Conference on Learning Representations (ICLR)*, volume 2026, pages 152171–152228.

Jinyang Zhang, Kexin Yang, Yu Wan, Muyang Ye, Baosong Yang, Fei Huang, Junyang Lin, and Dayiheng Liu. 2025. Nova-63: Native omni-lingual versatile assessments of 63 disciplines. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 7158–7200.

Yaowei Zheng, Richong Zhang, Junhao Zhang, YeYanhao, and Zheyang Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Association for Computational Linguistics (ACL)*, pages 400–410.

A Related Works

Schema Linking. Schema linking plays a pivotal role in text-to-SQL systems, aiming to align natural language queries with their corresponding tables and columns within a database. To achieve tighter integration between schema information and query semantics, several models—such as RAT-SQL (Guo et al., 2019), SchemaGNN (Bogin et al.,

2019), and ShadowGNN (Chen et al., 2021)—introduce relation-aware self-attention mechanisms to better capture structural dependencies. Building upon this direction, SADGA (Cai et al., 2021) proposes a dual-graph architecture that jointly encodes and aggregates the structural representations of natural language questions and database schemas.

With the emergence of large language models (LLMs) demonstrating remarkable performance across NLP tasks (Zhang et al., 2026a; Wang et al., 2026; Liao et al., 2026; Yang et al., 2026b), recent research has explored leveraging them for schema linking. For instance, MCS-SQL (Lee et al., 2025) employs multiple prompts and diverse response sampling from LLMs to eliminate irrelevant schema elements, thereby mitigating mismatches. TA-SQL (Qu et al., 2024a) performs schema linking by leveraging its Task Alignment strategy, which guides the LLM to utilize prior task experiences through in-context learning, thereby reducing hallucinations and improving the accuracy of linking natural language queries to the correct database schema elements. RSL-SQL (Cao et al., 2024) adopts a bidirectional schema linking strategy to more thoroughly exploit external knowledge from the database and to analyze how the generated SQL statements align with the elements necessary to accurately address user queries.

B Methodology Details

B.1 Notation Table

B.2 Algorithm Description

Algorithm 1 provides a step-by-step description of the SchemaToken framework, including both the Semantic Alignment and Pseudo-task Alignment stages.

B.3 Prompts in Two-Stage Training

In this section, we provide a detailed introduction to the prompts used in our framework.

Prompts for Data Augmentation in Semantic Alignment

1. Generate the description for the given database column. Only output the description without any additional text.
2. What is the meaning of this database column? Output only the meaning without any extra information.

Datasets	#Domains	#DB	#Table/DB	#Column/DB	Train	Dev	Test	Schema Linking (Table)			Schema Linking (Column)		
								MAX	MIN	AVG	MAX	MIN	AVG
Spider (Yu et al., 2018)	138	200	5.1	27.1	7862	1831	2147	4	1	1.5	9	1	2.8
BIRD (Li et al., 2023b)	37	95	7.3	54.2	9428	1534	1789	4	1	1.8	12	1	4.4
Spider 2.0 (Lei et al., 2025)	213	152	51.7	812.1	/	/	120	31	1	2.6	124	2	9.5

Table 4: Overview of datasets used in our experiments.

Symbol	Description
q	Natural language query
S	Database schema
t_i	The i -th table in the schema
C_i	Set of columns belonging to table t_i
c_{ij}	The j -th column of table t_i
$\hat{N}$	Number of tables in the database
$\hat{M}_i$	Number of columns in table t_i
C^*	Ground-truth set of schema-linked columns
f_θ	Schema linking function parameterized by θ
θ	Parameters of the large language model (LLM)
τ	Schema token
e_τ	Embedding of schema token τ
B_T	Table codebook
B_C	Column codebook
b_n^T	n -th code in the table codebook
b_n^C	n -th code in the column codebook
L_T, L_C	Number of quantization levels for tables and columns
$q_T^{(l)}(\cdot)$	l -th table quantization function
$q_C^{(l)}(\cdot)$	l -th column quantization function
h_t	Table representation before quantization
h_c	Column representation before quantization
$\mathcal{V}_{\text{orig}}$	Original LLM vocabulary
$\mathcal{V}_{\text{schema}}$	Set of schema tokens
$\mathcal{V}'$	Expanded vocabulary
h	Hidden state of the final transformer layer
d	Hidden dimension of the LLM
$\mathbf{W}', \mathbf{b}'$	Parameters of the extended LM_head
x	Input instruction for semantic alignment
y	Target semantic output
$\mathcal{L}_{\text{Sem}}$	Semantic regularization loss
$\mathcal{L}_{\text{Div}}$	Diversity regularization loss
α	Weight for diversity regularization
$\mathcal{L}_{\text{align}}$	Total alignment loss
B	Batch size
K	Number of clusters in k -means
$\mathcal{G}_k$	The k -th cluster of code embeddings
e_c^i	Nearest code embedding for schema token τ
e_+	Positive code embedding from the same cluster
e_j	Code embedding from a different cluster
π_θ	LLM policy in reinforcement learning
q'	Pseudo-task instruction
m	Number of schema tokens in a pseudo-task
$\tilde{e}, \hat{e}$	Lower and upper bounds of m
$\hat{C}$	Predicted set of schema tokens
C_{true}	Ground-truth set of schema tokens
$J(\cdot, \cdot)$	Jaccard similarity function
$R(\hat{C}, C_{\text{true}})$	Reward function for schema linking

Table 5: Notation Table used in SchemaToken.

3. Provide an explanation for this database column. Only return the explanation without any additional context.
4. Explain the function of this database column. Output just the description of the column

without any extra text.

5. What does this column represent in the database? Output only the description.
6. Interpret the meaning of this database column. Only provide the meaning without extra text.
7. What is the description of this database column? Output only the description without any extra text.
8. Can you provide a description for this database column? Please output only the description without any additional information.
9. What does this column in the database signify? Only output the description.
10. Explain the meaning of this database column. Only answer with the meaning without any additional text.

Prompts for Data Augmentation in Pseudo-task Alignment

1. Identify the database columns related to the following descriptions: {desc_list}.
2. Given these semantic definitions, return the corresponding schema tokens: {desc_list}.
3. Which database schema tokens match the following meanings? {desc_list}.
4. Map the provided descriptions to their exact schema tokens: {desc_list}.
5. Extract the relevant schema tokens that represent these functions: {desc_list}.
6. Translate the following column descriptions into their corresponding schema tokens: {desc_list}.
7. Find the database schema tokens that align with the following semantics: {desc_list}.
8. Please output the schema tokens that correspond to these column descriptions: {desc_list}.
9. Retrieve the specific schema tokens representing the following database attributes: {desc_list}.
10. Determine the correct schema tokens for the database fields described here: {desc_list}.

C Experimental Setups

C.1 Network Architecture

Our experiments were based on various backbone LLMs, including the Qwen3 series (Bai et al., 2023) of various sizes (Qwen3-1.7B and Qwen3-8B) and LLaMA3.1-8B-Instruct (Touvron et al., 2023). For Spider 2.0, where small-parameter models achieve near-zero SQL generation performance, we adopt the respective backbone model as the generator for schema linking and use GPT-4 as the Text-to-SQL generation model.

C.2 Datasets

To evaluate the effectiveness of SchemaToken, we conducted experiments on three publicly available Text-to-SQL datasets: BIRD-dev (Li et al., 2023b), Spider-dev (Yu et al., 2018), and Spider 2.0 (Lei et al., 2025). Since the Spider-dev and Spider 2.0 datasets do not provide descriptive information for each column, and some columns in the BIRD-dev dataset also lack such information, we employed GPT-4 (Achiam et al., 2023) to assist in generating these missing descriptions.

- **BIRD** (Li et al., 2023b) is a comprehensive large-scale benchmark developed for evaluating Text-to-SQL models in realistic, cross-domain environments. It consists of 12,751 natural language questions paired with their corresponding SQL queries, distributed over 95 databases from 37 distinct domains. The dataset is partitioned into 9,428 samples for training, 1,534 for development, and 1,789 for testing. By emphasizing large and diverse databases as well as the integration of external knowledge, BIRD presents a highly demanding setting that closely reflects practical Text-to-SQL challenges.
- **Spider** (Yu et al., 2018) is another widely used cross-domain benchmark for Text-to-SQL research. It comprises 10,181 natural language questions and 5,693 distinct SQL queries covering 200 databases across 138 different domains. Two evaluation protocols are commonly adopted: (i) *example split*, where 7,862/1,831/2,147 questions are randomly allocated to train/dev/test splits, permitting the same database to appear in multiple partitions; and (ii) *database split*, where 206 databases are divided into 130/36/40 for train/dev/test, ensuring each database is exclusive to a single split. Owing to its abundance of complex SQL structures—including joins, nested

queries, groupings, and orderings—Spider imposes substantial challenges for cross-domain generalization to previously unseen database schemas.

- **Spider 2.0** (Lei et al., 2025) (in our work, we use the Spider 2.0-snow version) is a large-scale, cross-domain Text-to-SQL benchmark that significantly scales up in database size, query complexity, and SQL dialect diversity compared to its predecessor. It comprises 120 test examples across 213 databases, with an average of 812.1 columns per database and 161.8 tokens per SQL query, reflecting a high degree of schema complexity and query length. Each SQL query involves an average of 6.8 functions, indicating extensive use of advanced SQL constructs.

C.3 Training Data Construction

We construct the training data for the two alignment stages only from database schema metadata. For Semantic Alignment, each training instance consists of a schema token paired with the corresponding column name and column description. BIRD provides semantic descriptions for most columns, while Spider-dev and Spider 2.0-snow do not provide complete column-level descriptions. For missing descriptions, we use GPT-4 to generate concise descriptions based on the table name, column name, and other available schema context, and we use the prompt templates in Appendix B.3 to augment the surface form of the instruction.

For Pseudo-task Alignment, training instances are synthesized online from the target database schema. We randomly sample a set of m columns, where $m \in [\tilde{\epsilon}, \hat{\epsilon}]$, concatenate their descriptions into the pseudo-task instruction, and use the corresponding unordered set of schema tokens as the ground-truth target for reward computation. The model is therefore optimized to recover the correct schema-token set from semantic descriptions rather than to memorize benchmark questions.

Importantly, neither stage uses user queries, gold SQL annotations, or gold schema-linking labels from the evaluation examples. The training data are limited to schema metadata and automatically synthesized pseudo tasks, whereas evaluation is performed on natural-language questions paired with gold SQL or schema-linking labels. This separation avoids train-test leakage, including for Spider 2.0-snow, which is primarily used as an evaluation benchmark in our experiments.

C.4 Evaluation Metrics

C.4.1 Schema Linking

Non-Strict Recall (NSR) Previous works have rarely provided an explicit definition of recall in the context of schema linking, despite its essential role in determining the fidelity of SQL generation. To fill this gap, we formally define the recall metric used in this study.

The *Non-Strict Recall* (NSR) quantifies the proportion of relevant schema elements that are correctly retrieved by the linking process. Formally, it is defined as the total number of correctly linked schema elements (i.e., intersection between predicted and ground-truth schema sets) divided by the total number of ground-truth schema elements across all questions:

$$\text{NSR} = \frac{\sum_{i=1}^n |S_{\text{gt},i} \cap \tilde{S}_i|}{\sum_{i=1}^n |S_{\text{gt},i}|} \quad (11)$$

where n denotes the total number of questions, $S_{\text{gt},i}$ represents the ground-truth schema elements for the i -th question, and $\tilde{S}_i$ refers to the schema elements predicted by the linking model.

Non-Strict Precision (NSP) Analogous to NSR, we further introduce the *Non-Strict Precision* (NSP), which evaluates how many of the linked schema elements are indeed correct. It measures the alignment between the predicted schema and the ground truth from the perspective of precision. The definition is as follows:

$$\text{NSP} = \frac{\sum_{i=1}^n |S_{\text{gt},i} \cap \tilde{S}_i|}{\sum_{i=1}^n |\tilde{S}_i|} \quad (12)$$

A higher NSP indicates that the schema linking process introduces fewer redundant or irrelevant columns, contributing to more concise and accurate SQL generation.

Non-Strict F1 (NSF) To provide a balanced evaluation that considers both recall and precision, we define the *Non-Strict F1* (NSF) score as the harmonic mean of NSR and NSP:

$$\text{NSF} = \frac{2 \times \text{NSR} \times \text{NSP}}{\text{NSR} + \text{NSP}} \quad (13)$$

This metric jointly reflects the overall effectiveness of schema linking in terms of both completeness and correctness.

Strict Recall Rate (SRR) In addition to the soft evaluation above, we adopt a stricter criterion to assess schema recall. Specifically, if the predicted schema set $\tilde{S}_i$ fully covers all schema elements required to generate the correct SQL for question i , we assign a value of 1; otherwise, 0. The *Strict Recall Rate* (SRR) is defined as the proportion of questions satisfying this condition:

$$\text{SRR} = \frac{\sum_{i=1}^n \mathbb{I}(\tilde{S}_i \supseteq S_{\text{gt},i})}{n} \quad (14)$$

Our method is designed to maximize SRR while maintaining the size of $\tilde{S}$ as compact as possible to minimize the prompt length. SRR thus reflects the model’s ability to retrieve all necessary schema components for exact SQL reconstruction.

C.4.2 Text-to-SQL

Execution Accuracy (EX) Execution Accuracy (EX) measures the proportion of SQL queries whose execution results precisely match those of the ground-truth SQL queries. It is reported as the percentage of correctly executed queries in the evaluation dataset, serving as the ultimate indicator of end-to-end task success.

C.5 Baselines

We adopted several state-of-the-art approaches as our compared baselines, including RESDSQL (Li et al., 2023a), CodeS (Li et al., 2024), MCS-SQL (Lee et al., 2025), TA-SQL (Qu et al., 2024a), and RSL-SQL (Cao et al., 2024). It is worth noting that RESDSQL and CodeS adopt locally customized small models to perform schema linking, whereas MCS-SQL, TA-SQL, and RSL-SQL originally employed GPT-4 as the backbone LLM for schema linking. To ensure a fair comparison, we replaced their backbone LLMs with the same local LLM used in our experiments. In addition, we introduced a baseline prompt-based approach, which simply uses the column names and their corresponding descriptions to prompt the LLM to perform the schema linking task.

The details of the baseline schema linking algorithms we used are as follows:

1. **RESDSQL** (Li et al., 2023a): RESDSQL decouples schema linking and skeleton parsing by integrating ranking-enhanced encoding and skeleton-aware decoding, reducing parsing difficulty for complex SQL queries.

2. **CodeS** (Li et al., 2024): CodeS is a series of pre-trained language models designed for Text-to-SQL tasks. It leverages incremental SQL-centric pre-training and bidirectional data augmentation.
3. **MCS-SQL** (Lee et al., 2025): MCS-SQL leverages multiple prompts to robustly refine schemas, generate diverse candidate SQL queries, and select the optimal query through a multiple-choice aggregation strategy.
4. **TA-SQL** (Qu et al., 2024a): TA-SQL mitigates LLM hallucinations by leveraging a Task Alignment (TA) strategy, guiding models to draw on experiences from similar tasks through in-context learning.
5. **RSL-SQL** (Cao et al., 2024): RSL-SQL enhances schema linking through bidirectional pruning, contextual information augmentation, and a binary selection mechanism, achieving high recall with reduced schema complexity.

C.6 Implementation Details

We used the PyTorch library to implement all algorithms based on the open-source HuggingFace transformers (Jain, 2022). We adopted LoRA for the second-stage policy adaptation, with the LoRA rank set to 8. In the Semantic Alignment stage, the original LLM is frozen and only the schema-token/codebook embeddings are trained for 3 epochs with a batch size of 64, using learning rates of 5×10^{-5} for Qwen3 models and 1×10^{-4} for LLaMA models. This stage was implemented using the LLaMAFactory framework (Zheng et al., 2024). For Pseudo-task Alignment, the schema-token/codebook embeddings are frozen, while the LoRA/adaptation parameters and the extended LM_{head} are optimized on our developed training framework for 200 steps, with a batch size of 1 and 4 rollouts per case. We set $\tilde{\epsilon}$ and $\hat{\epsilon}$ based on the statistical properties of each dataset (see Appendix C.2), while retaining a certain degree of extensibility. To further enhance the performance of SchemaToken, we integrated it with RSL-SQL (Cao et al., 2024), replacing the forward branch in RSL-SQL’s *Bidirectional Schema Linking* module with SchemaToken.

D Further Analysis and Supplementary Experiments

D.1 Hyper-Parameter Analysis

We further investigate several important hyper-parameters introduced in SchemaToken to facilitate future applications, including identifier length L and codebook size N in CodeBooks settings, strength of diversity regularization α and cluster K . The experimental results are shown in Fig. 6.

D.1.1 Identifier length L

We experiment with identifier lengths L ranging from 2 to 8, as shown in Fig. 6. The results indicate that 1) increasing L from 2 to 4 enhances performance, likely because very short identifiers may omit fine-grained information, reducing their expressiveness. 2) Further increasing L beyond 4 leads to a performance drop, which can be attributed to the error accumulation inherent in autoregressive generation (Ren et al., 2024). Since the correctness of each token in the sequence is crucial, generating longer identifiers accurately becomes more challenging than shorter ones.

D.1.2 CodeBook Size N

We assess SchemaToken with different codebook sizes $N = 64, 128, 256$, and 512. As presented in Fig. 6, 1) moderate increases in N generally improve performance, potentially due to enhanced diversity in code selection, which aids in distinguishing schema tokens. 2) However, excessively large codebooks may negatively affect results, as they can amplify noise in the semantic representation of schema tokens and potentially lead to overfitting of spurious patterns.

D.1.3 Strength of diversity regularization α

We vary the diversity regularization weight α from 0.001 to 0.1 and report the outcomes in Fig. 6. We observe that 1) increasing α tends to improve overall performance, reflecting the benefit of stronger diversity injection. Nonetheless, excessively large α may interfere with semantic regularization. 2) Empirically, we suggest setting $\alpha = 0.02$, as it provides a reasonable trade-off between semantic preservation and diversity, yielding optimal results.

D.1.4 Number of Clusters K

We investigate the impact of different cluster numbers K ranging from 3 to 10, as depicted in Fig. 6. The performance peaks around $K = 5$, with either smaller or larger values leading to a decline.

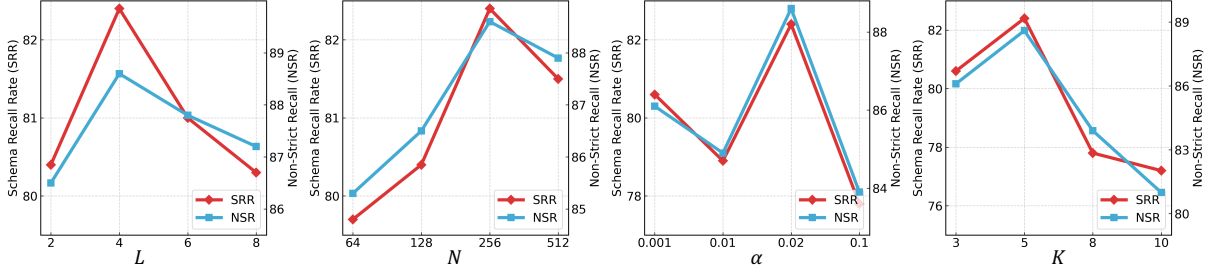


Figure 6: Performance of SchemaToken over different hyper-parameters on Spider-Dev.

This is intuitive: overly large clusters contain many code embeddings, reducing intra-cluster similarity, while very small clusters may over-constrain embeddings, causing them to be excessively similar within a cluster.

D.2 Codebook Diversity Analysis

To quantify the effect of the diversity regularization loss ($\mathcal{L}_{\text{Div}}$), we introduce a "utilization rate" metric, which measures the number of schema tokens assigned to each code in the codebook. We calculated the average, minimum, maximum, and standard deviation of this utilization rate across all codes, both with and without $\mathcal{L}_{\text{Div}}$.

Setting	Average	Min	Max	Std
w/ $\mathcal{L}_{\text{Div}}$	6.17	3	12	2.64
w/o $\mathcal{L}_{\text{Div}}$	6.17	1	56	32.52

Table 6: Codebook utilization rate with and without diversity regularization.

The quantitative results demonstrate that without regularization, a small fraction of the codes are heavily overused (max utilization of 56) while others are neglected (high standard deviation of 32.52). Incorporating $\mathcal{L}_{\text{Div}}$ ensures a highly balanced and healthy distribution of codebook usage, limiting the maximum utilization to 12 and reducing the standard deviation to 2.64.

To further empirically validate the superiority of the Codebook Diversity design, we conducted an analysis on the BIRD dev dataset. Based on our parameter sensitivity analysis, we adopted the optimal settings: Identifier length $L = 4$ and Codebook Size $N = 256$. We extracted and analyzed a total of 1936 columns across the dataset. As illustrated in Fig. 7, without the diversity regularization ($\mathcal{L}_{\text{Div}}$), the baseline method suffers from severe index collapse, where a small number of schema tokens are heavily overused while the vast

majority remain completely idle (frequency of 0). In contrast, SchemaToken incorporates the Codebook Diversity design, yielding a highly balanced code frequency histogram. Almost all available codes are effectively utilized, which significantly enhances the model’s representational capacity and proves the effectiveness of our diversity regularization strategy.

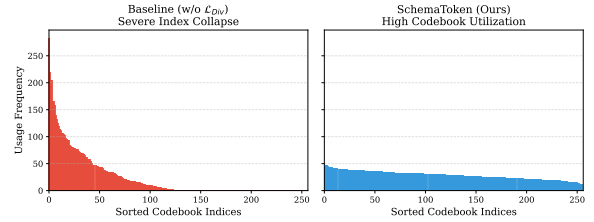


Figure 7: Code frequency histograms comparing the codebook utilization on the BIRD dev dataset (1936 columns, $L = 4$, $N = 256$). **Left:** The baseline method without diversity regularization suffers from severe index collapse. **Right:** SchemaToken incorporates $\mathcal{L}_{\text{Div}}$, yielding a highly diverse and balanced token utilization across the entire codebook.

D.3 Analysis under Different Database Scales

To further examine when SchemaToken is most beneficial, we divide Spider 2.0 databases into three groups according to the number of columns: small databases with fewer than 1K columns, medium databases with 1K–10K columns, and large databases with more than 10K columns. Table 7 reports the performance of different schema-linking methods under these scale groups.

The advantage of SchemaToken increases with database scale. On small databases, all methods can still access a relatively manageable schema context, yet SchemaToken remains the strongest overall, achieving 79.2 SRR, 84.6 NSR, and 66.0 NSF. On medium-scale databases, the gap becomes larger. On large databases with more than 10K columns, all baselines fail to execute any query cor-

Methods	Small (<1K cols, 76 DBs)					Medium (1K–10K cols, 53 DBs)					Large (>10K cols, 23 DBs)				
	Schema Linking				T2SQL	Schema Linking				T2SQL	Schema Linking				T2SQL
	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX
Prompt	60.8	78.5	44.6	56.9	20.8	44.6	65.4	23.7	34.8	20.2	0.0	4.6	45.0	8.3	0.0
MCS-SQL	66.3	80.4	46.8	59.2	25.6	44.2	68.4	26.7	38.4	20.2	0.0	13.3	24.5	17.2	0.0
TA-SQL	63.1	76.4	42.5	54.6	22.1	40.5	63.2	20.9	31.4	15.8	0.0	8.9	35.6	14.2	0.0
RSL-SQL	76.5	82.6	52.1	63.9	38.4	50.4	68.4	30.5	42.2	24.0	0.0	2.9	50.0	5.5	0.0
SchemaToken	79.2	84.6	54.1	66.0	40.6	65.7	74.8	36.8	49.3	26.9	47.8	65.8	30.0	41.2	11.6

Table 7: Performance on Spider 2.0 databases grouped by schema scale using Qwen3-8B as the backbone LLM.

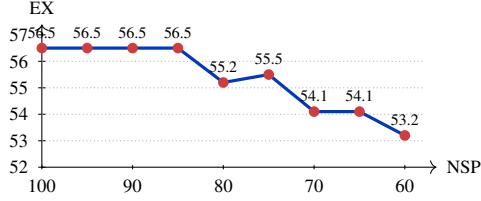


Figure 8: Impact of irrelevant-column noise on downstream Text-to-SQL execution accuracy on BIRD-dev with Qwen3-8B. The x-axis reports NSP when NSR is fixed at 100%.

rectly and obtain 0.0 SRR, whereas SchemaToken still achieves 47.8 SRR and 11.6 EX. This result supports our central motivation that schema tokenization targets a qualitatively different operating regime where context-heavy schema-linking methods become infeasible.

D.4 Impact of Reduced Precision

Although SchemaToken prioritizes recall and can introduce more irrelevant columns than precision-oriented methods, downstream Text-to-SQL generation is more sensitive to missing required columns than to a moderate number of extra columns. To quantify this effect, we start from the ground-truth schema-linking results on BIRD-dev with Qwen3-8B, progressively inject randomly selected irrelevant columns, and evaluate the resulting execution accuracy.

The results show that execution accuracy remains stable when NSP decreases from 100% to 85%, and degrades only mildly as more irrelevant columns are injected. In contrast, missing any required column prevents the SQL generator from constructing the correct query in most cases. This explains why optimizing SRR and NSR is particularly important for schema linking, while moderate precision degradation can be tolerated when the predicted schema set still covers the required columns.

D.5 RL Training Variance

To evaluate the stability of our reinforcement learning stage, we simulated multiple trials by running our experiments across 5 different random seeds. Table 8 reports the mean and standard deviation of these runs for SchemaToken and the strongest baseline, RSL-SQL. The low variance confirms the robustness of our method against random initialization and verifies the high reliability of our reported experimental results.

D.6 Analysis of Description Generator

In this section, we further analyze the impact of different description generators on SchemaToken. We consider GPT-4 (Achiam et al., 2023), DeepSeek-V3 (Liu et al., 2024a), and GLM-4-Plus (GLM et al., 2024) as alternative description generators. The experimental results are reported in Table 12.

The results show that although the performance of SchemaToken exhibits minor variations as the quality of the descriptions changes, its overall performance remains stable. More importantly, across all description generators, SchemaToken consistently outperforms the strongest baseline methods considered, providing further evidence of its robustness.

D.7 Analysis on Database Update Scenarios

We further investigate the effectiveness of SchemaToken under database update scenarios. We consider three representative cases:

1. **Column Insertion.** We assume an existing database (denoted as Ora. DB, the original database) and augment it with content from an updated database (denoted as Up. DB), i.e., expanding the tables (including columns) in Ora. DB. We set BIRD as Ora. DB and Spider as Up. DB, and also evaluate the reverse setting, with Spider as Ora. DB and BIRD as Up. DB.
2. **Column Deletion.** We randomly remove a sub-

Methods	Spider-dev					BIRD-dev					Spider 2.0				
	Schema Linking				T2SQL	Schema Linking				T2SQL	Schema Linking				T2SQL
	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX
<i>Qwen3-1.7B</i>															
RSL-SQL	77.4 $\pm$ 1.2	84.1 $\pm$ 1.2	39.6 $\pm$ 1.1	52.7 $\pm$ 1.4	63.8 $\pm$ 1.4	80.8 $\pm$ 1.2	93.0 $\pm$ 1.2	31.7 $\pm$ 1.0	45.7 $\pm$ 1.0	25.6 $\pm$ 1.0	53.2 $\pm$ 1.4	71.0 $\pm$ 0.8	15.6 $\pm$ 1.4	28.8 $\pm$ 1.2	21.9 $\pm$ 1.1
SchemaToken	81.4 $\pm$ 1.1	88.7 $\pm$ 1.2	58.0 $\pm$ 1.1	67.6 $\pm$ 0.9	67.5 $\pm$ 1.4	80.0 $\pm$ 1.0	94.5 $\pm$ 1.1	61.1 $\pm$ 1.1	75.1 $\pm$ 0.9	32.2 $\pm$ 0.9	61.9 $\pm$ 1.0	77.9 $\pm$ 1.0	35.5 $\pm$ 1.2	48.7 $\pm$ 1.1	26.7 $\pm$ 1.1
<i>Qwen3-8B</i>															
RSL-SQL	76.6 $\pm$ 1.3	83.5 $\pm$ 0.9	56.9 $\pm$ 0.9	69.5 $\pm$ 1.3	79.1 $\pm$ 1.3	80.6 $\pm$ 0.9	94.9 $\pm$ 1.4	48.2 $\pm$ 1.3	62.6 $\pm$ 1.2	49.0 $\pm$ 1.1	70.9 $\pm$ 1.0	85.5 $\pm$ 0.9	23.5 $\pm$ 0.9	33.1 $\pm$ 1.1	22.3 $\pm$ 1.4
SchemaToken	81.9 $\pm$ 1.5	89.7 $\pm$ 0.8	68.2 $\pm$ 1.2	77.6 $\pm$ 1.2	82.1 $\pm$ 1.2	85.8 $\pm$ 0.8	96.7 $\pm$ 1.1	74.9 $\pm$ 1.3	82.1 $\pm$ 0.9	51.8 $\pm$ 1.2	78.2 $\pm$ 1.0	84.8 $\pm$ 0.9	46.2 $\pm$ 0.9	57.3 $\pm$ 1.3	31.6 $\pm$ 1.3
<i>LLaMA3.1-8B</i>															
RSL-SQL	78.3 $\pm$ 0.9	87.3 $\pm$ 1.4	42.2 $\pm$ 0.9	57.5 $\pm$ 1.1	71.5 $\pm$ 1.5	88.0 $\pm$ 1.1	96.5 $\pm$ 1.0	34.2 $\pm$ 1.2	47.2 $\pm$ 1.1	42.4 $\pm$ 1.4	62.9 $\pm$ 1.3	78.0 $\pm$ 1.3	22.6 $\pm$ 1.1	33.7 $\pm$ 0.8	18.9 $\pm$ 1.1
SchemaToken	80.7 $\pm$ 1.1	89.9 $\pm$ 1.4	58.3 $\pm$ 0.9	70.1 $\pm$ 1.3	78.9 $\pm$ 1.0	92.0 $\pm$ 1.0	99.1 $\pm$ 0.9	49.5 $\pm$ 1.3	66.0 $\pm$ 0.9	49.3 $\pm$ 1.3	67.4 $\pm$ 1.2	82.9 $\pm$ 1.2	43.8 $\pm$ 1.4	60.2 $\pm$ 1.2	28.3 $\pm$ 0.9

Table 8: Mean and standard deviation of schema linking and Text-to-SQL performance across 5 random seeds.

set of columns from the database—while ensuring that columns required for subsequent schema linking tasks are preserved—to simulate this scenario. In this case, SchemaToken only needs to remove the corresponding tokens from the LLM’s schema token table.

3. **Column Renaming.** Since SchemaToken is independent of column names, no modification is required, allowing it to seamlessly adapt to this scenario.

The experimental results are summarized in Table. 9. Despite large-scale column insertions, SchemaToken consistently maintains robust performance and outperforms the strongest baseline methods considered, demonstrating its effectiveness under database update scenarios.

D.8 Analysis of Potential Solutions

In this section, we examine two potential approaches for mitigating context explosion, namely Table-to-Column Schema Linking and Retrieval-Augmented Generation (RAG).

1. **Direct Column Schema Linking vs. Table-to-Column Schema Linking.** We explore a two-stage strategy that first retrieves tables and then retrieves columns. The motivation is that table retrieval does not require column-level information and therefore remains within the LLM’s context budget; subsequently, column retrieval operates on a reduced set of tables, limiting the number of irrelevant columns introduced into the context. The experimental results are reported in Table. 10, where both strategies adopt the most basic prompting setup. The results show that Table-to-Column Schema Linking performs substantially worse than Direct Column Schema Linking. Further analysis reveals that, in the first stage, the lack of column-level information makes it difficult to accurately judge

table relevance based solely on table metadata, leading to a large number of relevant tables not being recalled. This failure propagates to the second stage, ultimately resulting in poor column recall.

2. **Retrieval-Augmented Generation (RAG).** We also investigate RAG-based strategies to address context explosion. We evaluate three mature RAG frameworks—RagFlow², Dify³, and LlamaIndex⁴—which are widely used in industrial-scale document retrieval. The experimental results, shown in Table. 11, indicate that all RAG frameworks exhibit poor performance and fall significantly behind SchemaToken. This performance gap can likewise be attributed to incomplete information modeling: RAG frameworks compute similarity between the user query and individual columns in isolation, while ignoring the global structure of the database. To the best of our knowledge, recent advances in Text-to-SQL research (Pourreza et al., 2025; Li et al., 2025; Lee et al., 2025; Qu et al., 2024b) consistently rely on LLMs reasoning over global database information for schema linking, and there has been little to no prior work adopting RAG as a viable solution for schema linking.

E Qualitative Analysis

E.1 Case Study

We present case studies to further illustrate the advantages of SchemaToken over traditional strategies. Specifically, we present the following two cases.

In Case#1, we observed that schema name-

²<https://ragflow.io/>

³<https://docs.dify.ai/en/use-dify/getting-started/introduction>

⁴<https://developers.llamaindex.ai/python/framework/>

Methods	Ori. DB	Up. DB	Spider-dev					BIRD-dev				
			Schema Linking				T2SQL	Schema Linking				T2SQL
			SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX
RSL-SQL	/	/	76.8	84.8	58.1	69.0	77.9	82.1	94.4	47.3	63.0	47.9
Scenario of Column Insertion												
SchemaToken	BIRD	Spider	83.4	89.0	62.1	73.2	80.9	83.9	96.1	75.1	84.3	49.8
SchemaToken	Spider	BIRD	81.7	85.4	64.0	73.2	79.9	84.5	96.8	72.1	82.6	52.2
Scenario of Column Deletion												
SchemaToken	/	/	81.9	86.4	63.4	73.1	79.9	85.1	95.4	71.1	81.5	51.8

Table 9: Analysis of SchemaToken under database update scenarios. Qwen3-8B is employed as backbone LLM. **Ori. DB:** Original Database. **Up. DB:** Updated Database.

Methods	Schema Linking				T2SQL
	SRR	NSR	NSP	NSF	EX
<i>Spider-dev</i>					
Direct Column Schema Linking	47.0	67.6	76.0	71.6	55.4
Table-to-Column Schema Linking	30.6	52.3	70.9	60.2	40.6
<i>BIRD-dev</i>					
Direct Column Schema Linking	10.5	45.4	74.0	56.3	18.4
Table-to-Column Schema Linking	7.2	32.8	66.4	43.9	11.9

Table 10: Direct Column Schema Linking vs. Table-to-Column Schema Linking. Qwen3-8B is employed as backbone LLM.

Methods	Schema Linking				T2SQL
	SRR	NSR	NSP	NSF	EX
<i>Spider-dev</i>					
RagFlow	54.3	70.5	44.3	54.4	57.6
Dify	51.4	68.5	41.4	51.6	52.4
LlamaIndex	49.8	60.2	46.7	52.6	53.7
SchemaToken	82.4	88.6	67.4	76.6	80.2
<i>BIRD-dev</i>					
RagFlow	24.0	57.3	32.1	41.1	22.3
Dify	26.7	59.8	30.4	40.3	20.6
LlamaIndex	22.6	51.5	34.8	41.5	22.1
SchemaToken	84.2	96.2	74.1	83.7	51.2

Table 11: Performance comparison of SchemaToken with various RAG solutions. Qwen3-8B is employed as backbone LLM.

based solutions are prone to errors when handling database fields containing spaces. As illustrated in the example above, when generating Enrollment (K-12) and Enrollment (Ages 5-17), the model outputted only Enrollment, failing to fully capture the meaning of the schema and mistakenly

treating Enrollment as a complete database field. SchemaToken, by treating the entire database field as an indivisible atomic unit, completely avoids this issue.

A similar observation arises in Case#2. We found that LLMs have weak understanding of the underscore “_” character and often output it as a hyphen “-”. We attribute this to the fact that both “_” and “-” occupy semantically negligible positions in the original LLM embedding space and are thus not easily distinguished. Moreover, “-” occurs more frequently than “_” in common text, biasing the LLM toward generating “-”. In database scenarios, however, “_” is very common, and frequent mis-generation of “_” greatly reduces the practicality of schema name-based solutions. Similar to Case#1, SchemaToken avoids this issue entirely by treating the database field as an atomic, indivisible unit.

Case#1

[question:] Consider the average difference between K-12 enrollment and 15-17 enrollment of schools that are locally funded, list the names and DOC type of schools which has a difference above this average.

[GT result:]
schools.School
schools.DOC
schools.CDSCode
schools.FundingType
frpm.CDSCode
frpm.Enrollment (K-12)
frpm.Enrollment (Ages 5-17)

[Schema Name result:]

Methods	Description Generator	Spider-dev					BIRD-dev				
		Schema Linking				T2SQL	Schema Linking				T2SQL
		SRR	NSR	NSP	NSF	EX	SRR	NSR	NSP	NSF	EX
Qwen3-1.7B											
RSL-SQL	/	77.3	85.4	38.2	52.8	64.2	79.2	93.6	31.3	46.9	27.3
SchemaToken	GLM-4-Plus	79.1	88.6	54.2	67.3	66.8	79.4	93.5	61.9	74.5	30.8
	DeepSeek-V3	81.2	89.0	52.6	66.1	68.4	81.5	96.6	58.5	72.9	32.9
	GPT-4	80.5	88.7	56.4	69.0	67.9	80.6	95.2	61.4	74.7	31.7
Qwen3-8B											
RSL-SQL	/	76.8	84.8	58.1	69.0	77.9	82.1	94.4	47.3	63.0	47.9
SchemaToken	GLM-4-Plus	91.5	85.2	64.5	73.4	79.4	83.5	96.0	76.5	85.1	49.5
	DeepSeek-V3	82.9	89.0	63.1	73.8	80.2	85.0	96.8	73.0	83.2	52.1
	GPT-4	82.4	88.6	67.4	76.6	80.2	84.2	96.2	74.1	83.7	51.2
LLaMA3.1-8B											
RSL-SQL	/	78.7	85.6	41.3	55.7	71.1	88.2	96.8	32.3	48.4	43.4
SchemaToken	GLM-4-Plus	81.9	87.3	59.7	70.9	77.0	89.6	96.3	51.3	66.9	47.5
	DeepSeek-V3	81.5	89.0	60.4	72.0	76.6	91.9	98.0	48.1	64.5	50.1
	GPT-4	82.7	89.4	58.6	70.8	77.8	90.2	97.1	49.1	65.2	48.6

Table 12: Robustness analysis of SchemaToken using various description generators.

schools.School schools.DOC schools.CDSCode schools.FundingType frpm.CDSCode frpm.Enrollment [SchemaToken result:] schools.School schools.DOC schools.CDSCode schools.FundingType frpm.CDSCode frpm.Enrollment (K-12) frpm.Enrollment (Ages 5-17)	superhero.superhero_name superhero.eye-colour_id superhero.hair-colour_id colour.id colour.colour [SchemaToken result:] superhero.superhero_name superhero.eye_colour_id superhero.hair_colour_id colour.id colour.colour
Case#2 [question:] Please list the superhero names of all the superheroes that have blue eyes and blond hair. [GT result:] superhero.superhero_name superhero.eye_colour_id superhero.hair_colour_id colour.id colour.colour [Schema Name result:]	Case Study (from BIRD datasets) Query: "What are the cards for set OGW? State the colour for these cards." Ground Truth: set_translations.setCode SchemaToken Output: cards.setCode Analysis: Both the cards table and the set_translations table contain a setCode column. The error arises because the user query itself is highly ambiguous. By loosely

using the word "cards" to describe the target information, the query introduces a strong surface-level lexical bias. This misleads the model into matching the literal word "cards" to the `cards.setCode` token, rather than inferring the true intent corresponding to `set_translations.setCode`.

Handling this type of ambiguity—where the user query is vaguely phrased, lacks specific constraints, and uses terminology that lexically overlaps with incorrect tables—is an inherent, long-standing challenge in the broader Text-to-SQL domain.

F Detailed Break-even Analysis: Prompt vs. SchemaToken

While SchemaToken incurs a one-time training cost (X), it achieves massive inference efficiency gains by eliminating verbose context. Let X denote the one-time training time, and let Y and Y' denote the inference latency per query for SchemaToken and the prompt-based baseline, respectively. For N queries, the total temporal cost for SchemaToken is $C_{ST} = X + N \cdot Y$, whereas the prompt-based approach costs $C_{Prompt} = N \cdot Y'$. Since $Y \ll Y'$, SchemaToken becomes strictly more cost-effective when the total number of queries exceeds the break-even point: $N > \frac{X}{Y' - Y}$.

To quantify this mathematically, we approximate the temporal cost using standard LLM serving throughput constraints (e.g., vLLM on a modern GPU like NVIDIA A100), assuming an input prefill speed of $\sim 2,000$ tokens/second and an output decode speed of ~ 40 tokens/second for an 8B model. For the prompt baseline (18.0k input, 128.5 output), the inference latency is roughly $Y' \approx \frac{18000}{2000} + \frac{128.5}{40} \approx 12.2$ seconds per query. In contrast, SchemaToken (1.5k input, 16.7 output) takes only $Y \approx \frac{1500}{2000} + \frac{16.7}{40} \approx 1.17$ seconds. Given the one-time training cost of $X = 4.4$ hours (15,840 seconds), solving the inequality $15840 + N \cdot 1.17 < N \cdot 12.2$ yields a break-even point of about **1.4K queries**.

This analysis clearly demonstrates that in production environments where thousands of queries are processed daily, SchemaToken is significantly more cost-effective and scalable than prompt-based methods. The one-time training overhead is amortized extremely quickly, resulting in an order-of-magnitude reduction in latency and a substantial decrease in memory consumption.

G Code and Data Availability

To support reproducibility and facilitate future research, we will publicly release all code and processed datasets upon publication. For reference and transparency, the complete code is also provided in <https://github.com/MrBlankness/SchemaToken>.

H Use of Large Language Models

In this work, LLMs are used in a supportive role to assist with language refinement and programming debugging tasks. All LLM-assisted outputs are carefully reviewed, verified, and revised by the authors before inclusion. The core research ideas, methodological design, experimental setup, and result analysis are conceived and carried out entirely by the authors.

Algorithm 1 Training Paradigm of SchemaToken

Require: Database schema $\mathcal{S}$, Codebook size N , levels L_T, L_C , LLM θ , Diversity weight α , RL learning rate η .

Ensure: Trained LLM parameters θ and codebook embeddings $\mathcal{B}_T, \mathcal{B}_C$.

- 1: **// Stage 1: Semantic Alignment**
 - 2: Initialize $\mathcal{B}_T$ and $\mathcal{B}_C$ randomly.
 - 3: Freeze original LLM parameters θ_{orig} and train only the schema-token/codebook embeddings $\mathcal{B}_T, \mathcal{B}_C$.
 - 4: **for** each semantic instruction x and target description y **do**
 - 5: Construct schema token embedding $\mathbf{e}_\tau = \sum_{l=1}^{L_T} q_T^{(l)}(\mathbf{h}_t) + \sum_{l=1}^{L_C} q_C^{(l)}(\mathbf{h}_c)$.
 - 6: Compute Semantic Loss $\mathcal{L}_{\text{Sem}} = -\sum_t \log p_\theta(y_t | y_{<t}, x)$.
 - 7: Compute Diversity Loss $\mathcal{L}_{\text{Div}}$ using K -means clustering.
 - 8: Update schema-token/codebook embeddings using $\nabla(\mathcal{L}_{\text{Sem}} + \alpha \mathcal{L}_{\text{Div}})$.
 - 9: **end for**
 - 10: **// Stage 2: Pseudo-task Alignment**
 - 11: Freeze $\mathcal{B}_T, \mathcal{B}_C$, and update LoRA/adaptation parameters and LM_head'.
 - 12: **for** each pseudo-task instruction q' and ground-truth schema set C_{true} **do**
 - 13: Generate predicted set of schema tokens $\hat{C} \sim \pi_\theta(\cdot | q')$.
 - 14: Compute Jaccard reward $R = |\hat{C} \cap C_{\text{true}}| / |\hat{C} \cup C_{\text{true}}|$.
 - 15: Update the trainable adaptation parameters and LM_head' via GRPO to maximize expected reward $\mathbb{E}[R]$.
 - 16: **end for**
 - 17: **return** $\theta, \mathcal{B}_T, \mathcal{B}_C$
-