LitE-SQL: A Lightweight and Efficient Text-to-SQL Framework with Vector-based Schema Linking and Execution-Guided Self-Correction

Shengmin Piao[*], **Jieun Lee**[*], **Sanghyun Park**[†]

Yonsei University

Seoul, South Korea

{shengminp, jieun199624, sanghyun}@yonsei.ac.kr

Abstract

The Text-to-SQL task translates natural language questions into SQL queries, enabling intuitive database interaction for non-experts. While recent methods leveraging Large Language Models (LLMs) achieve strong performance, their reliance on proprietary models raises concerns about deployment feasibility and data privacy. In this work, we introduce LitE-SQL, a Lightweight and Efficient framework with two components: (i) a Schema Retriever that performs efficient schema linking using a vector database of pre-computed schema embeddings, optimized with a hard-negative supervised contrastive objective to distinguish semantically similar but functionally irrelevant columns, and (ii) a SQL Generator fine-tuned in two stages—supervised fine-tuning followed by execution-guided reinforcement—enabling execution-guided self-correction without multi-candidate sampling, which is commonly required by prior LLM-based approaches. On BIRD, LitE-SQL achieves 72.10% execution accuracy, and on Spider 1.0 it reaches 88.45%, demonstrating comparable or superior performance to LLM-based methods despite using 2× to 30× fewer parameters. Our findings demonstrate that high-quality Text-to-SQL generation is feasible with lightweight models, offering a practical solution for privacy-sensitive and resource-constrained settings. Code is available at https://github.com/shengminp/LitE-SQL.

1 Introduction

Relational databases remain a cornerstone of modern data management, with Structured Query Language (SQL) serving as the standard interface for querying structured data. However, composing SQL queries poses a barrier for non-expert users. This has motivated growing interest in Text-to-SQL, which translates natural language questions into SQL queries, enabling intuitive interaction with relational databases (Hong et al., 2025; Li et al., 2024b).

Early approaches relied on rule-based systems (Li and Jagadish, 2014), whereas advances in deep neural networks substantially improved semantic parsing and SQL generation (Li et al., 2023). Despite these gains, existing methods still struggle with cross-domain generalization and complex query generation (Singh et al., 2025). Recently, Large Language Models (LLMs) have demonstrated remarkable performance via in-context learning capabilities (Brown et al., 2020). These approaches enhance SQL generation by designing structured prompts and incorporating database schema information into the input context (Li et al., 2023; Pourreza and Rafiei, 2023; Gao et al., 2024; Li et al., 2024a; Lee et al., 2025; Talaei et al., 2024; Maamari et al., 2024; Pourreza et al., 2025a).

Although LLM-based methods mitigate several limitations of earlier approaches, they introduce new concerns—particularly data privacy and deployment constraints arising from reliance on proprietary models (Hong et al., 2025) such as ChatGPT (Achiam et al., 2023) or Gemini (Team et al., 2023). This reliance risks information leakage in domains with strict data security requirements. Developing custom LLMs is a possible remedy but incurs substantial development and maintenance costs. These limitations underscore the need for lightweight and efficient alternatives.

To this end, we identify two core technical challenges: **(i) Contextual Expansion of Schema Information.** Incorporating complete schema information into model inputs is critical for accurate generation but substantially increases input length and, due to the quadratic complexity of the self-attention mechanism (Vaswani et al., 2017), results in high memory consumption. Such long-context inputs impose a considerable computational burden on lightweight models, hindering efficient pro-

---

[*] Same contribution.

[†] Corresponding author.

cessing in resource-constrained environments. **(ii) Computational Footprint of Large-Parameter LLMs.** While LLM-based methods achieve strong performance in SQL generation, each inference incurs significant computational cost, which is often exacerbated by strategies such as multi-candidate generation. These limitations highlight the need for lightweight alternatives that offer competitive performance while remaining feasible in constrained computational settings.

To address these challenges, we propose **LitE-SQL**, a **Li**ghtweigh**t** and **E**fficient framework comprising a *Schema Retriever* and a *SQL Generator*. We design a retrieval-based schema linking approach using a vector database. Unlike existing methods that rely on LLMs (Pourreza et al., 2025a; Talaei et al., 2024) or heavyweight encoders (Li et al., 2024b), our Schema Retriever stores precomputed schema embeddings in a vector database and, at inference, encodes only the question to retrieve relevant columns through semantic similarity search. To our knowledge, this is the first work to fully exploit a vector-database-driven retriever for schema linking in Text-to-SQL. For the SQL Generator, we develop a two-stage fine-tuning strategy that tackles computational cost by employing lightweight language models. In the first stage, the model generates SQL conditioned on the question and retrieved schema; in the second stage, it refines the outputs using execution feedback. This execution-guided self-correction enables efficient error correction without relying on multi-candidate generation.

We conduct comprehensive experiments on two benchmark datasets, BIRD (Li et al., 2023) and Spider 1.0 (Yu et al., 2018), to evaluate the effectiveness of LitE-SQL. On BIRD, LitE-SQL achieves an execution accuracy of 72.10%, and on Spider 1.0, it reaches 88.45%. These results are competitive with LLM-based approaches while using approximately 2× to 30× fewer parameters, highlighting its efficiency and suitability for resource-constrained environments. Furthermore, the Schema Retriever achieves retrieval quality on par with or superior to existing schema linking approaches, and the SQL Generator consistently corrects erroneous SQL queries using execution feedback, as evidenced by a measurable reduction in error rates in self-correction analyses. These findings indicate that LitE-SQL offers a practical and effective lightweight framework for Text-to-SQL.

The key contributions of this work are as follows:

- We identify core challenges in deploying lightweight models for Text-to-SQL and propose LitE-SQL, a lightweight and efficient framework consisting of a Schema Retriever and a SQL Generator.

- We introduce a hard-negative supervised contrastive learning objective tailored for schema-level retrieval in Text-to-SQL.

- We develop a two-stage fine-tuning strategy for the SQL Generator, enabling lightweight models to self-correct with execution feedback without multi-candidate sampling.

2 Related Work

2.1 Schema Linking

Schema linking identifies relevant schema elements for a given question, a key step toward generating correct SQL. Early methods trained classifiers to rank schema elements (Li et al., 2024b; Pourreza and Rafiei, 2024), relying on supervised filters to score column and table relevance.

Recent approaches leverage LLMs through in-context learning, injecting full or partial schema information into prompts (Gao et al., 2024; Li et al., 2025; Wang et al., 2025). Some adopt multi-step prompting—such as generating dummy SQL or asking models to justify schema selections—to identify relevant columns (Qu et al., 2024; Lee et al., 2025). CHESS (Talaei et al., 2024) introduces a multi-agent framework where an Information Retriever collects context and a Schema Selector identifies relevant schema elements. Distillery (Maamari et al., 2024) questions the need for explicit schema linking, showing that strong LLMs can infer schema implicitly. However, smaller models degrade significantly without filtering irrelevant columns, underscoring the importance of schema selection for lightweight models.

Despite their effectiveness, these methods incur substantial overhead by re-encoding the schema for every input. In addition, CHESS and CHASE-SQL employ embedding similarity via vector databases, but mainly for keyword retrieval, leaving final schema selection to LLMs. In contrast, we leverage vector databases as the core retriever for schema linking, explicitly designed for lightweight Text-to-SQL models, while encoding only the input question at inference. To further exploit vector

databases effectively, we optimize the embedding space with a tailored objective, improving retrieval quality and overall Text-to-SQL performance.

2.2 SQL Generation

SQL generation methods with language models fall broadly into in-context learning and fine-tuning.

In-context learning. LLMs demonstrate strong SQL generation capabilities when guided by carefully designed prompts. A basic strategy provides the database schema alongside an instruction to generate the query (Li et al., 2023). Few-shot prompting further improves performance by adding few examples (Pourreza and Rafiei, 2023; Maamari et al., 2024; Li et al., 2024a). To handle complex questions, structured prompts decompose questions into sub-questions (Pourreza and Rafiei, 2023; Wang et al., 2025; Pourreza et al., 2025a), or employ chain-of-thought prompting (Wei et al., 2022), which generates SQL queries incrementally(Li et al., 2023; Wang et al., 2025; Pourreza et al., 2025a). Many methods further apply self-consistency (Wang et al., 2023), generating multiple candidates and selecting the majority. While effective, these strategies are computationally expensive due to multiple LLM calls per question. In contrast, our work explores fine-tuned lightweight models that avoid such overhead while preserving competitive accuracy.

Fine-tuning. Fine-tuning pre-trained models provides stronger domain adaptation, fine-grained control, and better suitability for privacy-sensitive deployments. Given that powerful models such as GPT-4 (Achiam et al., 2023) remain proprietary, recent efforts emphasize open-source models fine-tuned for SQL generation (Li et al., 2024b, 2025; Pourreza and Rafiei, 2024; Pourreza et al., 2025b). To enhance the understanding between natural language and SQL, fine-tuning with a language modeling objective on SQL data offers a direct and effective way. Building on this foundation, methods such as data augmentation (Li et al., 2025), incremental pretraining (Li et al., 2024b), and task decomposition (Pourreza and Rafiei, 2024; Pourreza et al., 2025b) further enhance performance. Nevertheless, both fine-tuning and these extensions still often generate incomplete or invalid SQL queries. To address this limitation, we incorporate reinforcement learning with execution feedback, enabling self-correction and improving query validity.

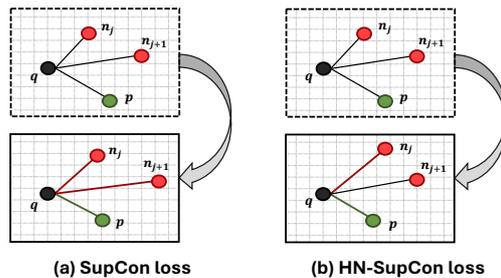(a) SupCon loss   (b) HN-SupCon loss

Figure 1: Conceptual illustration of the difference between supervised contrastive (SupCon) loss and hard-negative filtered supervised contrastive (HN-SupCon) loss.

3 Methodology

3.1 Schema Retriever

To enable efficient schema linking with lightweight models, we adopt a retrieval-based approach in which candidate columns are retrieved from a vector database.

Each column is encoded into a dense embedding derived from its schema metadata (column name, description, table name, and value description). These embeddings are pre-computed and indexed in a vector database. At inference time, the question is embedded and compared against stored columns to retrieve the top-k candidates by cosine similarity.

Effective retrieval requires not only general semantic similarity but also precise alignment between the question and the relevant columns. We therefore fine-tune the embedding model with a supervised contrastive objective. Unlike the standard supervised contrastive (SupCon) loss (Khosla et al., 2020), which contrasts positives with all defined negatives based on semantic similarity, our setting further distinguishes columns that are semantically similar but irrelevant for SQL generation.

We observed that pre-trained models already assign low similarity to clearly irrelevant columns, making them easy negatives. The main challenge lies in hard negatives—columns that appear semantically related but are not required for SQL generation. To address this, we introduce a hard-negative filtered supervised contrastive loss (HN-SupCon), illustrated in Figure 1. Inspired by recent work that masks semantically related in-batch samples to avoid false negatives (Zhang et al., 2025), HN-SupCon selectively updates only hard negatives and ignores trivial ones.

Formally, given a question embedding q_i, its

positive column p_i, and negatives $\{n_{ij}\}_j^{N_i}$, the loss is defined as:

$$\mathcal{L}_{\text{HN-SupCon}} = -\frac{1}{B}\sum_i^B \log\left(\frac{e^{s(q_i,p_i)/\tau}}{Z_i}\right) \quad (1)$$

$$Z_i = e^{s(q_i,p_i)/\tau} + \sum_j^{N_i} m_{ij}\, e^{s(q_i,n_{ij})/\tau} \quad (2)$$

where $s(\cdot,\cdot)$ denotes cosine similarity and τ is a temperature parameter. The mask m_{ij} filters easy negatives:

$$m_{ij} = \begin{cases} 1 & \text{if } q_i \odot n_{ij} \geq q_i \odot p_i - 0.1, \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The margin 0.1 is chosen empirically (Table 8 in Appendix). For efficiency, we limit the number of negatives sampled per question, as large schema (e.g., in BIRD) contains many irrelevant columns (See Appendix B.3).

3.2 SQL Generator

Supervised Fine-tuning (SFT). To enhance the SQL generation ability of the pre-trained model (θ), we first train the model to learn a conditional mapping from a natural language question (Q) and schema information (S) to the corresponding SQL query (SQL). The schema information includes column data types, primary key constraints, descriptive metadata, representative samples, and foreign key relations. The training objective is:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\log P(SQL \mid Q, S; \theta) \quad (4)$$

During inference, the model utilizes a fixed number k of schema information retrieved by the Schema Retriever. However, training examples often contain fewer than k schema information. To ensure consistency between training and inference, we augment training inputs by including only the schema elements explicitly referenced in the gold SQL, and filling the remaining slots with randomly sampled irrelevant schema information until reaching the fixed length k (Figure 2). This approach allows the model to learn SQL generation in the presence of irrelevant but plausible schema information, which simulates potential false positives returned by the retriever at inference time while maintaining a consistent input structure. The value of k is determined empirically based on dataset characteristics (Section 5.3).
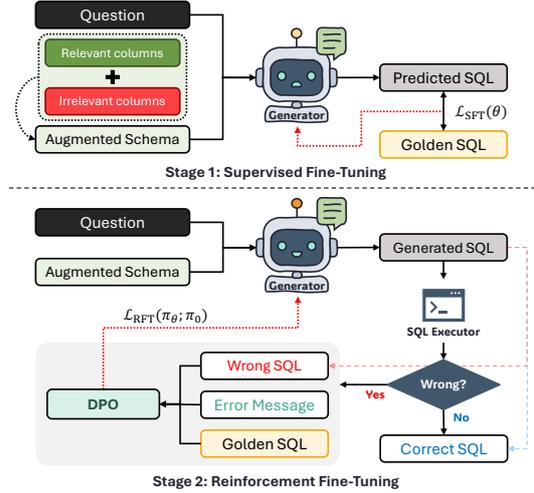

Figure 2: Two-stage training strategy of SQL Generator.

Reinforcement Fine-tuning (RFT). After the model reliably produces syntactically valid SQL, we train it to revise outputs using execution feedback. Specifically, we employ Direct Preference Optimization (DPO) (Rafailov et al., 2023) to align model behavior with preference signals derived from query execution results.

DPO operates on pairwise comparisons $\mathcal{D} = \{x_i, y_i^w, y_i^l\}_{i=1}^N$, where x_i is the input, and y_i^w and y_i^l represent two generated queries with y^w preferred over y^l. Instead of human-annotated preference, we generate multiple candidate queries per instance and label them by execution: successfully executing queries are preferred over failed ones. For failed queries, we also collect feedback (error messages) and incorporate it into the inputs used during SFT as illustrated in Figure 2.

To ensure data quality, when only one failed query exists, the corresponding ground truth query is treated as the preferred output. If multiple failed cases are present, we randomly sample non-overlapping successful outputs as preferred, ensuring the ground truth appears at least once. When successful queries are insufficient, we replicate the ground truth query to maintain balance.

Given the constructed dataset, the model is optimized using the following training objective:

$$\begin{aligned}
&\mathcal{L}_{\text{RFT}}(\pi_\theta; \pi_0) \\
&= \mathcal{L}_{\text{DPO}}(y_i^w, y_i^l | x_i) + \alpha \mathcal{L}_{\text{NLL}}(y_i^w | x_i) \\
&= -\log \sigma \left(\beta \log \frac{\pi_\theta(y_i^w|x_i)}{\pi_0(y_i^w|x_i)} - \beta \log \frac{\pi_\theta(y_i^l|x_i)}{\pi_0(y_i^l|x_i)} \right) \\
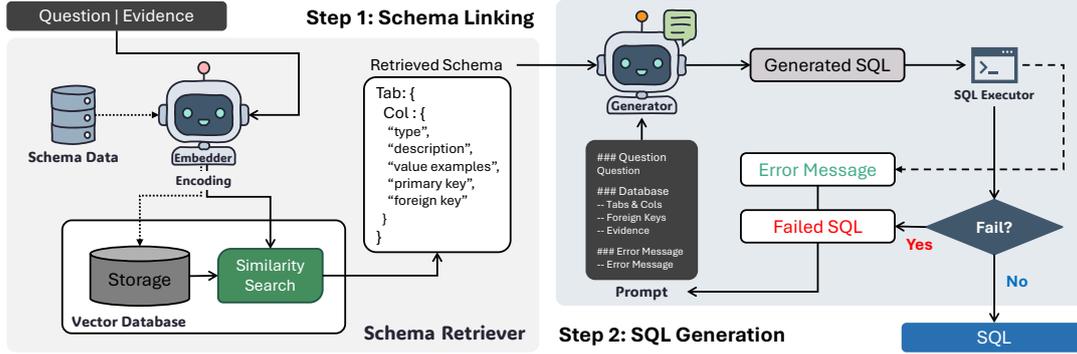&\quad - \alpha \log \pi_\theta(y_i^w|x_i)
\end{aligned}$$

$$(5)$$

Figure 3: The inference pipeline of LitE-SQL.

where π_0 is a fixed reference model initialized from π_θ. The hyperparameter α controls the weight of the negative log-likelihood loss, and β regulates divergence of π_θ from π_0.

3.3 Inference

We now describe how LitE-SQL operates at inference time (Figure 3). We assume preprocessing is completed: (i) database schemas are embedded and stored in a vector database using the fine-tuned embedding model (Embedder), and (ii) the SQL Generator has been trained through the two-stage process.

Given a user question (optionally with evidence)[1], the input is encoded by Embedder and compared with stored schema embeddings via cosine similarity. The Schema Retriever then retrieves the top-k candidate columns together with column types, descriptions, value examples, and key indicators (e.g., primary/foreign key status). The SQL Generator produces an initial SQL query conditioned on the question and retrieved schema, which is then executed against the target database.

If execution fails, the system gathers feedback—including failed SQL query, corresponding error message, original question, and retrieved schema context—and feeds it back to the model for revision. During training, this execution feedback is incorporated into the model input and used to construct preference pairs for optimization. At inference time, the same feedback signal is used iteratively for execution-guided self-correction, with the detailed input format shown in Table 14. The process repeats until a valid SQL query is produced or a predefined maximum number of iterations is reached, after which the final executable SQL is returned.

[1]Evidence annotations are unavailable in Spider 1.0.

4 Experimental Setup

4.1 Dataset

We evaluate our model on two widely-used Text-to-SQL benchmarks: BIRD (Li et al., 2023) and Spider 1.0 (Yu et al., 2018), both designed for cross-domain generalization. Specifically, Spider 1.0 includes 200 databases spanning 138 domains, while BIRD covers 95 large-scale databases from 37 professional domains. Both benchmarks feature complex SQL queries that involve operations such as JOIN, GROUP BY, and HAVING. Detailed dataset descriptions are provided in Appendix A.1.

4.2 Baselines

We compare LitE-SQL against several recent methods on BIRD and Spider 1.0, covering both in-context learning and fine-tuning paradigms. Full baseline details are provided in Appendix A.2.

4.3 Metrics

Execution Accuracy (EX). Following the standard in BIRD and Spider 1.0, we adopt EX as the primary evaluation metric. EX measures whether the predicted SQL query yields the same execution result as the ground-truth query on the target database.

$$\text{EX} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(V_n = \hat{V}_n), \qquad (6)$$

where V_n denotes the result set returned by executing the ground-truth query, and $\hat{V}_n$ is the result set from the predicted query. The indicator function $\mathbb{I}(\cdot)$ is defined as:

$$\mathbb{I}(V, \hat{V}) = \begin{cases} 1, & \text{if } V = \hat{V}, \\ 0, & \text{otherwise}. \end{cases} \qquad (7)$$

	Method	Generator	Size	BIRD (Dev) EX($\uparrow$)	Spider 1.0 (Test) EX ($\uparrow$)
In-context Learning	ChatGPT + CoT (Li et al., 2023)	GPT-3.5	175B	36.64	-
	DIN-SQL (Pourreza and Rafiei, 2023)	GPT-4	175B	50.72	85.30
	DAIL-SQL (Gao et al., 2024)	GPT-4	175B	54.76	86.60
	TA-SQL (Qu et al., 2024)	GPT-4	175B	56.19	-
	SuperSQL (Li et al., 2024a)	GPT-4	175B	58.50	-
	MAC-SQL (Wang et al., 2025)	GPT-4	175B	59.39	86.75
	MCS-SQL (Lee et al., 2025)	GPT-4	175B	63.36	**89.60**
	CHESS (Talaei et al., 2024)	Gemini 1.5 Pro	200B	68.31	87.20
	Distillery (Maamari et al., 2024)	GPT-4o	200B	67.21	-
	CHASE-SQL (Pourreza et al., 2025a)	Gemini 1.5	200B	**73.01**	87.60
Fine-tuning	CodeS (Li et al., 2024b)	StarCoder	7B	57.17	-
	OmniSQL (Li et al., 2025)	Qwen2.5-Coder	7B	63.90	87.90
	DTS-SQL (Pourreza and Rafiei, 2024)	DeepSeek	7B	55.80	77.10
	Reasoning-SQL (Pourreza et al., 2025b)	Qwen2.5-Coder	7B	68.05	78.72
	LitE-SQL	Qwen2.5-Coder	7B	<u>72.10</u>	<u>88.45</u>

The performance of the baselines is obtained from the corresponding paper.

Table 1: EX (%) of our model with correct schema and comparison with baselines on BIRD development set and Spider 1.0 test set. Among fine-tuning–based baselines with 7B parameters, LitE-SQL achieves the best performance. **Bold** numbers indicate the best performance, and <u>underlined</u> numbers indicate the second-best.

True Positive Rate (TPR). The proportion of ground-truth relevant columns that are retrieved by the Schema Retriever. Higher values indicate more effective retrieval of necessary columns.

$$\text{TPR} = \frac{\text{\# of retrieved relevant columns}}{\text{\# of ground-truth relevant columns}} \quad (8)$$

False Positive Rate (FPR). The proportion of retrieved columns that are not part of the ground-truth relevant columns. Lower values indicate better precision in schema retrieval.

$$\text{FPR} = \frac{\text{\# of retrieved irrelevant columns}}{\text{\# of retrieved columns}} \quad (9)$$

Schema Linking Recall (SLR). The proportion of queries where all relevant columns are successfully retrieved. We consider a case successful if the Schema Retriever retrieves the complete set of ground-truth relevant columns. Higher SLR indicates stronger schema linking performance.

$$\text{SLR} = \frac{\text{\# of successful cases}}{\text{\# of total queries}} \quad (10)$$

4.4 Implementation Details

We employ ChromaDB[2] as the vector database for lightweight integration, in-memory retrieval, and cosine similarity search. The embedding backbone for Schema Retriever is Qwen3-0.6B-Embedding (Zhang et al., 2025), fine-tuned with the HN-SupCon loss (Section 3.1). We adopt

Qwen2.5-Coder models (Hui et al., 2024) with 1.5B, 3B, and 7B parameters for SQL Generator. All models are fine-tuned on 4 A100 GPUs with the Huggingface library (Wolf, 2019), initialized from publicly released checkpoints[3]. Detailed hyperparameter settings are provided in Appendix A.3.

5 Results and Analysis

5.1 Baselines Comparison

We evaluate LitE-SQL on the BIRD development set[4] and the Spider 1.0 test set. Recent Text-to-SQL systems typically perform schema linking in advance, using LLMs to extract relevant schema information. To ensure a controlled comparison, we adopt the same setting and provide the resulting schema information during SQL generation.

We first compare LitE-SQL against in-context learning methods. As shown in Table 1, most methods rely on 175B-200B (Shittu, 2024) parameter models, whereas our fine-tuned 7B model achieves 72.10% on BIRD and 88.45% on Spider 1.0, outperforming the majority of GPT-4 and Gemini-1.5 based methods. Although CHASE-SQL surpasses our method by 0.91% on BIRD, we outperform it by 0.85% on Spider 1.0. Compared to MCS-SQL, which achieves the highest Spider 1.0 score, our method yields a 1.15% gap while exceeding it by 8.74% on BIRD. These results underscore the generalization capability of LitE-SQL, offering

---

Retriever	Generator		EX (↑)	
	SFT	RFT	BIRD (Dev)	Spider 1.0 (Test)
✗	✗	✗	39.31	61.61
✓	✗	✗	43.16 (+3.85)	64.28 (+2.67)
✓	✓	✗	58.21 (+18.90)	83.56 (+21.95)
✓	✓	✓	**60.56 (+21.25)**	**84.35 (+22.74)**

Table 2: Overall result of ablation study.

Method	Size	TPR (↑)	FPR (↓)	SLR (↑)	EX (↑)
CodeS (Li et al., 2024b)	3.5B	89.64	74.16	65.31	51.70
CHESS (Talaei et al., 2024)	200B	87.15	**8.27**	61.9	**57.14**
LitE-SQL	0.6B	**95.23**	80.28	**82.31**	56.46

Table 3: Performance of schema linking on subsampled-BIRD compared to baselines.

Model	TPR (↑)	FPR (↓)	SLR (↑)
Qwen3-Embedding-0.6B	82.80	83.62	59.91
+ SupCon	95.03	81.04	82.46
+ HN-SupCon	**96.71**	**80.71**	**87.42**

Table 4: Effect of loss functions on schema retrieval performance on BIRD development set.

Model	TPR (↑)	FPR (↓)	SLR (↑)
Qwen3-Embedding-0.6B	99.22	84.08	96.88
+ SupCon	96.32	84.12	96.13
+ HN-SupCon	**99.27**	**83.96**	**99.16**

Table 5: Effect of loss functions on schema retrieval performance on Spider 1.0 test set.

competitive performance with significantly fewer parameters.

Within the fine-tuning category, LitE-SQL consistently outperforms prior 7B-scale baselines, with average improvements of 10.87% on BIRD and 7.21% on Spider 1.0. The consistent improvements across benchmarks demonstrate that the two-stage training strategy efficiently utilizes model capacity, balancing predictive performance and computational efficiency.

5.2 Ablation Study

To evaluate the contributions of each component, we conduct a comprehensive ablation study, with results presented in Table 2 [5].

We first assess the effect of the Schema Retriever. In the baseline configuration, the retriever is disabled, and the complete database schema is fed directly to the SQL Generator without fine-tuning. This leads to a notable performance degradation compared to the retriever-enabled setup, highlighting the importance of filtering irrelevant schema elements to constrain the generator's search space, thereby underscoring the necessity of schema linking.

We then evaluate the two-stage training strategy. After the SFT stage, the model outperforms the prompt-engineering baseline by more effectively leveraging retrieved schema information for SQL generation. Importantly, this improvement should not be interpreted as diminishing the role of retrieval. Instead, SFT serves as a prerequisite that enables the model to fully benefit from schema retrieval. As shown in Table 3, once the genera-

---
[5] Unless otherwise specified, all reported results in subsequent sections are based on the 7B model.

tor has been sufficiently trained, retrieval quality becomes a decisive factor. For instance, CodeS with weaker schema linking imposes a clear performance ceiling, limiting the effectiveness of the generator.

Additional gains are achieved from the RFT stage, where execution feedback is used to revise SQL outputs. Although improvements over SFT are modest, RFT enhances robustness by reducing both semantic and syntactic errors. Detailed error analysis is provided in §5.4.

Overall, the ablation results reveal a clear stage-dependent interaction among schema retrieval, SFT, and RFT. SFT delivers the primary performance gains, schema retrieval constrains the generator's search space once training is in place, and RFT further improves robustness. Their combination produces cumulative improvements on both BIRD and Spider 1.0, validating the complementary roles of all components in the proposed framework.

5.3 Analysis of Schema Retriever

Schema linking baselines. In this experiment, we evaluate schema linking performance against representative baselines. For fairness, we use each method's schema linking strategy to extract relevant columns, while employing our SQL Generator for execution. Following prior work (Talaei et al., 2024; Maamari et al., 2024), evaluation is conducted on a subsampled BIRD development set (10% per database, 147 examples). We compare against CodeS and CHESS, two widely adopted baselines (Maamari et al., 2024; Pourreza et al., 2025a,b; Li et al., 2024a).

As shown in Table 3, SLR plays a crucial role in determining EX. Despite a higher FPR than CodeS, LitE-SQL's stronger SLR yields a 4.76% gap in EX, showing that higher recall can compensate for mod-
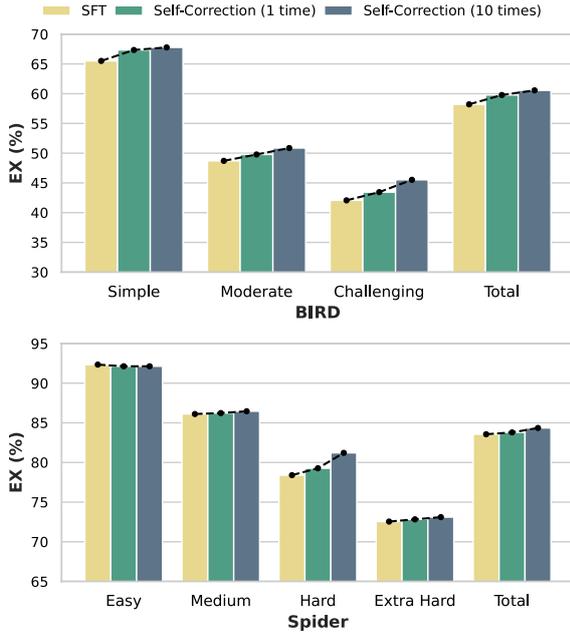
Figure 4: Effect of the number of self-correction iterations across different question difficulty.
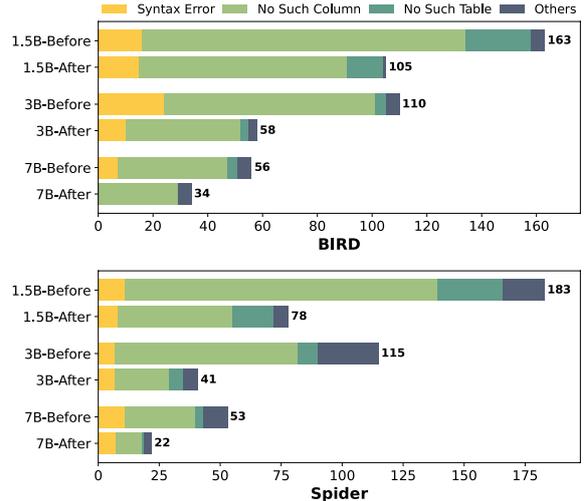

Figure 5: The distribution of execution error types before and after self-correction for each model size. The x-axis indicates the number of errors, and the number of each bar denotes the total errors.

erate false positives. By contrast, when the FPR gap is large, as between CHESS and LitE-SQL, excessive irrelevant columns reduce EX. The EX difference between CHESS and LitE-SQL is modest, only a 0.68% gap, yet notable given their difference in model scale and architecture: CHESS relies on a 200B-parameter multi-agent LLM with chain-of-thought reasoning, while LitE-SQL achieves comparable accuracy with a 0.6B-parameter retriever.

Effect of loss functions. We evaluate the impact of different training objectives on schema retrieval by comparing three settings: the vanilla embedding model (Qwen3), fine-tuned with SupCon loss (Qwen3 + SupCon), and fine-tuned with our proposed HN-SupCon loss (Qwen3 + HN-SupCon). As shown in Table 4 and Table 5, HN-SupCon achieves the best trade-off, yielding higher TPR and SLR while reducing FPR. These results confirm that emphasizing hard negatives enables the model to distinguish columns that are semantically similar but functionally irrelevant, thereby mitigating false positives and improving retrieval quality.

Effect of hyperparameters. We analyze the impact of two key hyperparameters in fine-tuning the embedding and filtering models: the number of negative samples and the number of retrieved columns, denoted as N_i and k, respectively. Because large schemas contain many irrelevant columns, we limit the number of negatives sampled per question. On

both datasets, setting $N_i = 8$ yields the best trade-off between FPR and SLR, while using $k = 25$ retrieved columns achieves the highest SLR. Detailed results of corresponding analysis on datasets are provided in Appendix B.3.

5.4 Analysis of SQL Generator

Effect of self-correction iterations. We evaluate the effectiveness of self-correction by analyzing both the number of correction iterations and their impact across different question difficulty levels, as illustrated in Figure 4. Our results demonstrate that incorporating executor feedback during generation consistently enhances EX across all correction rounds. This confirms the utility of the self-correction ability acquired during the RFT stage. Furthermore, the first self-correction pass yields the largest accuracy gain on the BIRD benchmark. While subsequent passes continue to improve performance, the gains diminish progressively, suggesting that most correctable errors are addressed early, with later iterations refining a small subset of harder cases.

Effect of execution feedback. To assess how the generator leverages executor feedback during self-correction, we categorize execution errors into four types: *syntax errors*, *no such column*, *no such table*, and *other* (miscellaneous errors). We compare the distribution of these error categories before and after self-correction to quantify improvements. As shown in Figure 5, self-correction consistently reduces the frequency of all error categories across

3600

model scales.

Semantic errors—particularly *no such column* and *no such table*—are of notable importance, as they reflect misalignment between the natural language question and the underlying database schema. Among these, *no such column* error, initially the most prevalent, decreases substantially after correction. Moreover, *no such table* errors are almost entirely eliminated in larger models. These results suggest that self-correction significantly improves semantic alignment, with larger models benefiting the most. *Syntax errors* are observed across all model sizes prior to correction but are reduced afterward, with the 7B model achieving full resolution on the BIRD dataset.

Overall, these findings underscore the effectiveness of the RFT stage in enhancing model robustness by mitigating both semantic and syntactic errors.

6   Conclusion

In this paper, we present LitE-SQL, a lightweight and efficient framework for Text-to-SQL with vector-based schema linking and execution-guided self-correction. We evaluated LitE-SQL on two representative datasets, showing that it matches or outperforms LLM-based baselines in EX while requiring up to $30\times$ fewer parameters. Beyond empirical gains, LitE-SQL can serve as a privacy-preserving and resource-efficient solution suitable for on-premises deployment.

Limitations

While LitE-SQL achieves competitive performance, there remain two areas for improvement.

First, retrieving a fixed k columns inherently increases the likelihood of including true positives, but at the same time inevitably introduces false positives. While our design partially mitigates this trade-off through embedding space optimization, reducing residual false positives remains a promising avenue for future improvement.

Second, while the current self-correction mechanism improves robustness by handling execution failures, it is mainly effective for syntactic errors that can be detected during execution. This design already mitigates a large portion of query failures, but semantically inaccurate yet executable queries—for instance, those joining the wrong table but still returning results—remain unresolved. Developing mechanisms that capture such semantic

errors represents a natural extension of our framework.

Acknowledgments

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*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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. *Proc. VLDB Endow.*, 17(5):1132–1145.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673.

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 *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353, Abu Dhabi, UAE.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to sql: Are we fully ready? *Proc. VLDB Endow.*, 17(11):3318–3331.

Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proc. VLDB Endow.*, 18(11):4695–4709.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024b. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.

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. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025a. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *The Thirteenth International Conference on Learning Representations*.

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*, 36:36339–36348.

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed text-to-SQL with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan O Arik. 2025b. Reasoning-SQL: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-SQL. In *Second Conference on Language Modeling*.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before generation, align it! a novel and effective strategy for

mitigating hallucinations in text-to-SQL generation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 5456–5471, Bangkok, Thailand.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

Hassan Shittu. 2024. Google gemini pro 1.5: All you need to know about this near perfect ai model. Fello AI Post.

Aditi Singh, Akash Shetty, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. A survey of large language model-based generative ai for text-to-sql: Benchmarks, applications, use cases, and challenges. In *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 00015–00021.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *CoRR*, abs/2405.16755.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557, Abu Dhabi, UAE.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

T Wolf. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

A   Detailed Experimental Setup

A.1   Datasets

BIRD[6] The BIRD benchmark aims to bridge the gap between academic research and real-world applications by encompassing 37 professional domains that present practical challenges, including noisy or incomplete database values, external knowledge integration, and SQL efficiency on large databases. It comprises questions categorized by difficulty: simple, moderate, and challenging, each paired with a SQL query, and annotated external knowledge for selected samples.

Spider 1.0[7] Spider 1.0 is a large-scale, cross-domain Text-to-SQL benchmark designed to evaluate the generalization ability of models across diverse databases and query structures. Each question is paired with an SQL query and categorized into four difficulty levels: easy, medium, hard, and extra hard, depending on query complexity. The benchmark emphasizes model performance on unseen databases and SQL queries, aiming to assess generalization beyond pattern memorization.

Table 6 summarizes key statistics of both datasets.

A.2   Baselines

In-context Learning. Prompt engineering is central to this paradigm. DAIL-SQL (Gao et al., 2024) selects examples by similarity to enhance relevance, while SuperSQL (Li et al., 2024a) introduces intermediate representations to improve semantic alignment. Distillery (Maamari et al., 2024) demonstrates effective augmentation and correction when schema fit within the context window.

---

[6]https://bird-bench.github.io/
[7]https://yale-lily.github.io/spider

Dataset	# DB	# Table/DB	# Row/DB
BIRD	95	7.3	549K
Spider 1.0	200	5.1	2K
Dataset	**# Train**	**# Valid**	**# Test**
BIRD	9376	1534	-
Spider 1.0	8659	1034	2147

Table 6: Database Statistics of BIRD and Spider 1.0.

To handle complex questions, several models employ question decomposition. DIN-SQL (Pourreza and Rafiei, 2023), MAC-SQL (Wang et al., 2025), and CHASE-SQL (Pourreza et al., 2025a) decompose questions into sub-questions, while MAC-SQL, ChatGPT + CoT (Li et al., 2023), and CHASE-SQL further integrate chain-of-thought prompting to enhance interpretability and accuracy. Pipeline-based approaches such as TA-SQL (Qu et al., 2024) and MCS-SQL (Lee et al., 2025) explicitly perform schema linking to reduce errors and improve schema recall. Multi-agent framework, like MAC-SQL, DIN-SQL, CHESS (Talaei et al., 2024), CHASE-SQL, and MCS-SQL, which modularize tasks such as schema linking, decomposition, and SQL refinement.

Fine-tuning. Fine-tuning language models on SQL corpora improves alignment between natural language and SQL. From a data perspective, CodeS (Li et al., 2024b) incrementally pre-trains on curated SQL datasets with schema-aware retrieval, while OmniSQL (Li et al., 2025) synthesizes SynSQL-2.5M, a large and diverse corpus designed for realistic query patterns.

Regarding training methodologies, DTS-SQL (Pourreza and Rafiei, 2024) implements a two-stage fine-tuning process that first handles schema linking before SQL generation. Reasoning-SQL (Pourreza et al., 2025b) integrates partial rewards such as schema linking accuracy and execution correctness via Group Relative Policy Optimization to foster intermediate reasoning.

A.3   Hyperparameters

The hyperparameter configurations for the implementation are summarized as follows.

Schema Retriever. The embedding model is trained for 1 epoch using the AdamW optimizer with a learning rate of 5×10^{-5} and temperature 0.07. Batch sizes are set to 16 for BIRD and 24 for Spider.

SQL Generator (SFT). Each model is trained for 2000 steps with a batch size of 16 using

Method	Generator	Size	BIRD (Dev) EX(↑)	Spider 1.0 (Test) EX (↑)
CodeS (Li et al., 2024b)	StarCoder	1B	50.46	-
		3B	55.02	-
		7B	57.17	-
		15B	58.47	-
OmniSQL (Li et al., 2025)	Qwen2.5-Coder	7B	63.90	87.90
		14B	64.20	<u>88.30</u>
		32B	64.50	87.60
DTS-SQL (Pourreza and Rafiei, 2024)	DeepSeek	7B	55.80	77.10
Reasoning-SQL (Pourreza et al., 2025b)	Qwen2.5-Coder	7B	68.05	78.72
		14B	**72.29**	81.43
LitE-SQL	Qwen2.5-Coder	1.5B	65.19	83.00
		3B	70.01	85.28
		7B	<u>72.10</u>	**88.45**

The performance of the baselines is obtained from the corresponding paper.

Table 7: Extended results for the fine-tuning baselines in Table 1.

Margin Values	BIRD (Dev)			Spider 1.0 (Test)		
	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)
0	96.08	80.83	85.46	99.18	83.98	98.88
0.1	**96.71**	**80.71**	**87.42**	**99.27**	**83.96**	**99.16**
0.2	96.22	80.81	86.18	99.03	83.98	98.74

Table 8: Evaluation of margin values for HN-SupCon loss.

Method	Avg.(s)	Std.(s)
CHESS (Talaei et al., 2024)	83.631	43.017
OmniSQL (Li et al., 2025)	76.759	1.002
LitE-SQL	**24.979**	31.414

Table 9: Latency on subsampled-BIRD compared to baselines.

the AdamW optimizer. Learning rates are set to 1×10^{-4} for the 1.5B and 3B models, and 5×10^{-5} for the 7B model. A cosine learning rate scheduler with a warm-up ratio of 0.05 is employed. We apply LoRA adaptation (Hu et al., 2022) with rank $r = 64$ for the 1.5B models, and $r = 128$ for the 3B and 7B models. The scaling factor is set as $\alpha_{\text{LoRA}} = 2r$, and dropout is fixed at 0.05.

SQL Generator (RFT). Each model is further fine-tuned for 2 epochs using pairwise preference samples generated with a temperature of 0.7. The batch size remains 16. Learning rates are set to 5×10^{-6} for all models. We again use the cosine scheduler with a warm-up ratio of 0.1. The weight β and α are set as 0.1 and 1.0. LoRA settings are consistent with those in the supervised fine-tuning.

B Additional Experiments

B.1 Extended Baselines Comparison

Table 7 presents an extended comparison between LitE-SQL and fine-tuned baseline models with various parameter counts. As shown in the table, several prior approaches rely on significantly larger models—up to 14B or 32B parameters—to achieve competitive performance. In contrast, LitE-SQL

attains comparable or superior accuracy with much smaller models. Notably, the 7B LitE-SQL model matches or outperforms larger variants of OmniSQL (14B and 32B) and Reasoning-SQL (14B) on both BIRD and Spider 1.0, despite using far fewer parameters. Overall, these results demonstrate that LitE-SQL achieves strong performance under a markedly reduced model budget, highlighting its superior parameter efficiency relative to existing fine-tuning approaches.

B.2 Efficiency Analysis

We evaluate the computational efficiency of LitE-SQL by measuring inference latency. We compare against two representative baselines that follow distinct learning paradigms: CHESS (in-context learning) and OmniSQL (fine-tuning with a 7B-parameter model). For all methods, latency is measured on a per-instance basis, and the reported mean and standard deviation are computed over all instances in subsampled-BIRD.

As shown in Table 9, both CHESS and OmniSQL incur substantial inference latency, with average per-instance runtimes of 83.631s and 76.759s, respectively. In contrast, LitE-SQL achieves sig-

nificantly lower latency, reducing the average inference time to 24.979s. This improvement reflects the efficiency of the proposed pipeline, which avoids repeated large-model invocations during schema selection and eliminates the need for generating multiple candidates. Overall, the results demonstrate that LitE-SQL substantially reduces inference-time overhead compared to representative in-context learning and fine-tuning baselines, highlighting its practical efficiency at the system level.

B.3 Hyperparameter Analysis

We report supplementary experiments that guided the choice of hyperparameters for the Schema Retriever. We analyze three hyperparameters: margin values for HN-SupCon loss, the number of negative samples for training (N_i), and the number of retrieved columns (k).

Margin values. We investigate the margin hyperparameter in the HN-SupCon loss, which defines the threshold for distinguishing hard negatives. Table 8 reports results obtained under the configuration of eight negative samples and top-25 retrieval columns, consistent with the configurations of N_i and k analyzed below. A margin of 0.1 provides the best trade-off, yielding the highest SLR while maintaining low FPR. Smaller margins (e.g., 0) fail to sufficiently separate hard negatives, whereas larger margins (e.g., 0.2) relax the criterion and reduce recall. Since performance already degrades at 0.2, further increasing the margin is expected to exacerbate this trend; we therefore restrict our analysis to $\{0, 0.1, 0.2\}$ and fix the margin to 0.1 in all experiments.

Negative sample sizes. The number of negative samples (N_i) affects both training efficiency and schema linking performance. We experimented with values from 3 to 8, with results reported in Tables 10 and 11. Across both datasets, larger sample sizes generally improved SLR by exposing the model to more challenging negatives, but gains saturated beyond $N_i = 5$. In particular, $N_i = 8$ consistently achieved the highest SLR under the top-25 setting (87.42 on BIRD and 99.16 on Spider), while maintaining a balanced trade-off with FPR. Adding more negatives would mostly introduce easy cases with limited contribution, while also increasing memory cost. We therefore fix $N_i = 8$ in all experiments.

Retrieved Columns. The number of retrieved columns provided to the SQL Generator has a direct impact on EX. We empirically evaluate top-$k \in \{10, 15, 20, 25\}$ with SQL Generators of 1.5B, 3B, and 7B parameters. Results are summarized in Table 12. On BIRD, top-25 consistently yields the best EX across all model sizes, reflecting the dataset's large and complex schemas, which require retrieving more columns to ensure coverage of relevant ones. In contrast, Spider contains smaller schemas and most queries involve only 1–4 relevant columns, so retrieving beyond top-15 introduces additional noise without substantial benefit. This effect is most evident for smaller models (1.5B and 3B), which are less robust to irrelevant columns. The 7B model, however, is better able to disregard noise and thus achieves peak performance at top-25 on both datasets. Accordingly, we set $k = 25$ in our final configuration.

C Input Templates

C.1 SQL Generator

For all datasets, we use the same input template for prompt design as illustrated in Table 13 and Table 14.

C.2 Schema Retriever

We provide the input templates—query and schema-document formats—used for training the embedding model (Tables 15 and 16).

Models	NegLimit_3			NegLimit_4			NegLimit_5		
	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)
Top10	86.19	62.36	60.43	87.05	61.97	62.52	87.51	61.58	61.34
Top15	91.48	72.12	71.25	92.13	71.92	74.32	92.83	71.66	75.68
Top20	94.34	77.49	78.75	94.54	77.48	80.77	95.32	77.26	82.79
Top25	95.90	80.87	83.64	96.04	80.85	85.14	96.69	80.72	87.29
Models	NegLimit_6			NegLimit_7			NegLimit_8		
	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)
Top10	87.96	61.49	63.36	87.52	61.74	63.10	89.14	60.85	**66.69**
Top15	92.67	71.75	75.68	92.64	71.78	74.84	93.49	71.45	**78.36**
Top20	94.90	77.37	82.07	95.24	77.31	82.27	95.54	77.20	**83.83**
Top25	96.05	80.85	85.40	96.78	80.71	86.70	96.71	80.71	**87.42**

Table 10: Comparison of different numbers of negative samples and retrieved columns on BIRD development set.

Models	NegLimit_3			NegLimit_4			NegLimit_5		
	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)
Top10	95.00	71.38	90.50	94.86	71.42	89.85	95.06	71.34	90.03
Top15	96.77	79.28	95.62	96.76	79.32	95.16	96.81	79.29	95.58
Top20	97.70	82.65	97.44	97.82	82.65	97.21	97.87	82.63	97.58
Top25	98.67	83.99	98.46	98.91	83.98	98.70	98.95	83.97	98.88
Models	NegLimit_6			NegLimit_7			NegLimit_8		
	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)	TPR (↑)	FPR (↓)	SLR (↑)
Top10	94.93	71.43	90.13	94.74	71.52	89.01	95.15	71.34	**90.36**
Top15	96.87	79.31	95.34	96.70	79.31	95.34	97.11	79.28	**95.48**
Top20	97.82	82.66	97.25	97.84	82.64	97.39	98.20	82.62	**97.76**
Top25	98.82	83.99	98.65	98.89	83.98	98.79	99.27	83.96	**99.16**

Table 11: Comparison of different numbers of negative samples and retrieved columns on Spider 1.0 test set.

Sizes	BIRD (Dev)				Spider 1.0 (Test)				
1.5B	Simple	Moderate	Chall.	Overall	Easy	Medium	Hard	Extra	Overall
Best	73.51	54.31	46.90	65.19	92.13	85.88	77.54	71.15	83.00
Top10	56.00	35.99	31.72	47.65	88.72	79.23	68.68	64.43	76.57
Top15	58.38	39.01	31.72	50.00	88.30	**80.05**	**69.96**	65.55	**77.27**
Top20	58.16	39.87	32.41	50.20	**89.36**	79.46	69.55	63.87	76.90
Top25	**59.46**	**40.30**	**35.86**	**51.43**	89.15	78.76	69.11	64.99	76.67
3B	Simple	Moderate	Chall.	Overall	Easy	Medium	Hard	Extra	Overall
Best	76.32	63.15	51.72	70.01	94.89	88.10	78.83	74.23	85.28
Top10	58.16	46.77	37.24	52.74	90.21	**84.36**	74.73	64.99	80.34
Top15	59.78	49.78	43.45	55.22	**90.21**	84.25	**77.75**	65.27	**81.00**
Top20	**62.05**	51.51	40.69	56.84	89.79	84.01	74.95	62.75	79.79
Top25	61.95	**51.29**	**43.45**	**56.98**	89.15	83.20	74.08	61.90	78.99
7B	Simple	Moderate	Chall.	Overall	Easy	Medium	Hard	Extra	Overall
Best	78.59	63.58	57.93	72.10	95.74	91.60	82.51	78.99	88.45
Top10	62.38	42.46	40.00	54.24	**92.77**	85.76	77.75	70.31	83.00
Top15	64.86	48.92	43.45	58.02	92.55	86.00	80.13	72.27	83.88
Top20	66.59	50.00	47.59	59.78	92.34	86.35	79.70	72.55	83.93
Top25	**67.78**	**50.86**	**45.52**	**60.56**	92.13	**86.46**	**81.21**	**73.11**	**84.35**

Table 12: Effect of the number of retrieved columns (top-k) on EX with SQL Generators of 1.5B, 3B, and 7B parameters. Results are reported by question difficulty (BIRD: simple, moderate, Chall.; Spider: easy, medium, hard, extra) as well as Overall performance. Chall. denotes challenging, and Overall denotes the aggregate performance across all difficulty levels.

INPUT TEMPLATE OF SFT

```
### Question
{QUESTION}

### Database
– Tables and Columns
{COLUMN_SCHEMA_INFORMATION}

# Example:
disp.disp_id = {
  "type": "integer",
  "primary_key": true,
  "values": [9, 2],
  "description": "unique number of identifying this row of record",
  "comment": "disposition id"
}
...
```
– Foreign Keys
{FOREIGN_KEYS_RELATIONS}

```
# Example:
disp.disp_id = card.disp_id
...
```
– Evidence
{EVIDENCE}

Table 13: Input template of SFT stage with example for column schema and foreign keys.

INPUT TEMPLATE OF RFT

{INPUT TEMPLATE OF SFT}

{SQL GENERATED WRONG}

Error Message
{ERROR MESSAGE}

Table 14: Input template of RFT stage with generated wrong SQL and corresponding execution feedback.

INPUT TEMPLATE OF QUERY

Instruct:Given a natural language question, retrieve database column information passages used to generate SQL.
Query:{QUESTION} {EVIDENCE}

Table 15: Input format used for training the embedding model. The {EVIDENCE} field is optional and omitted when unavailable (e.g., Spider).

INPUT TEMPLATE OF DOCUMENT

table:{TABLE_NAME}
column:{COLUMN_NAME}
column_desc:{COLUMN_DESCRIPTION}
value_desc:{VALUE_DESCRIPTION}

Table 16: Input format for database schema documents used in training the embedding model and stored in the vector database. The column_desc and value_desc fields are optional and omitted when not available.