DCG-SQL: Enhancing In-Context Learning for Text-to-SQL with Deep Contextual Schema Link Graph

Jihyung Lee[1*], **Jin-Seop Lee**[1*], **Jaehoon Lee**[1], **YunSeok Choi**[2†], **Jee-Hyong Lee**[3†]
[1]Department of Artificial Intelligence, Sungkyunkwan University
[2]Department of Immersive Media Engineering, Sungkyunkwan University
[3]Department of Computer Science and Engineering, Sungkyunkwan University
{jjklle, wlstjq0602, hoon1223, ys.choi, john}@skku.edu

Abstract

Text-to-SQL, which translates a natural language question into an SQL query, has advanced with in-context learning of Large Language Models (LLMs). However, existing methods show little improvement in performance compared to randomly chosen demonstrations, and significant performance drops when smaller LLMs (e.g., Llama 3.1-8B) are used. This indicates that these methods heavily rely on the intrinsic capabilities of hyper-scaled LLMs, rather than effectively retrieving useful demonstrations. In this paper, we propose a novel approach for effectively retrieving demonstrations and generating SQL queries. We construct a Deep Contextual Schema Link Graph, which contains key information and semantic relationship between a question and its database schema items. This graph-based structure enables effective representation of Text-to-SQL samples and retrieval of useful demonstrations for in-context learning. Experimental results on the Spider benchmark demonstrate the effectiveness of our approach, showing consistent improvements in SQL generation performance and efficiency across both hyper-scaled LLMs and small LLMs. The code is available at https://github.com/jjklle/DCG-SQL.

1 Introduction

Text-to-SQL aims to generate SQL queries given a pair of a natural language question and a database schema. Traditional studies focused on how to train models that took a question and a schema as input to generate the target SQL (Wang et al., 2020; Qi et al., 2022; Li et al., 2023b). However, they required training an encoder-decoder model with billions of parameters, leading to high training costs and limited generalization. Recently, with the advance of Large Language Models (LLMs), in-context learning with LLMs has been applied to Text-to-SQL tasks due to its efficiency and strong generalization ability across various tasks (Rajkumar et al., 2022; Chang and Fosler-Lussier, 2023; Pourreza and Rafiei, 2023).

Methods	Retrieval	GPT-4	Llama 3.1-8B
ACT-SQL	✓	83.9	75.0
	✗	83.2	75.6
DAIL-SQL	✓	82.1	69.9
	✗	80.8	69.3

Table 1: Execution Accuracy (%) for Spider Dataset. All experiments are conducted with 4-shot setting. When ✗ is applied, the demonstrations are randomly choosen.

To maximize the capability of in-context learning of LLMs, it is necessary to provide relevant and useful demonstrations to generate correct SQL query (Luo et al., 2024). Most existing approaches represent each sample using a text embedding derived solely from the question, and select demonstrations based on the similarity of the question embeddings (Lee et al., 2025; Guo et al., 2024; Zhang et al., 2023a; Gao et al., 2024). However, as shown in Table 1, their performance shows no difference from that of randomly chosen demonstrations. This result implies that existing methods failed to maximize the capabilities of in-context learning due to their limitation in selecting appropriate demonstrations. This becomes more apparent when they are applied to smaller LLMs which have weaker in-context learning capabilities compared to hyper-scale LLMs (Wei et al., 2022). Since smaller LLMs rely more heavily on demonstrations, performance significantly drops, sometimes even falling below the performance with randomly selected demonstrations.

This is because they overlook DB schema when representing Text-to-SQL samples. Since DB schema determines the structure and semantics of SQL queries, it must be incorporated into the representation. In Text-to-SQL tasks, understanding a

[*]Equal contribution
[†]Corresponding author

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15397–15412

question inherently relies on the database schema. Therefore, when retrieving relevant demonstrations, both the question and the schema should be jointly considered and represented in a unified manner. encoded through a unified representation.

In this paper, we propose a novel Text-to-SQL method for in-context learning with **D**eep **C**ontextual Schema Link **G**raph-based Retrieval, as we called DCG-SQL. We develop a novel graph-based joint representation of both the question and the schema to retrieve demonstrations. However, developing a joint representation that effectively captures the contextual relationship between the question and the schema presents two challenges.

First, datasets are not usually available to learn how schema items are relevant to the question, a process commonly known as schema linking (Guo et al., 2019; Wang et al., 2020). As a result, some previous works (Qi et al., 2022; Li et al., 2023b; Zhang et al., 2023b) have attempted to establish links through simple word matching, but this approach is often too simplistic and fails to accurately capture the relationship between questions and schema items. A novel method is needed to effectively model the contextual relationships necessary for Text-to-SQL samples without annotations.

The second challenge is not all schema items in a database are necessary for effectively representing the given Text-to-SQL sample. If all schema elements are considered, irrelevant schema items could hinder the effective representation. If irrelevant schema items are included in representations, question tokens could be connected to unrelated schema items. They also make it difficult to distinguish between samples with different questions that share the same DB schema. Therefore, when representing Text-to-SQL samples for demonstration retrieval, it is also essential to exclude irrelevant schema items.

To identify only relevant schema items to the question in a sample, we train a transformer based encoder with classification heads. Since each training sample contains a question, a schema and a target SQL, we can train the classifier with the question and the schema as input, and the SQL to indicate relevant schema items. When training the encoder, we add special tokens, such as [TAB] and [COL], to generate appropriate representations even when their names are composed of several words. These special tokens help to effectively capture their semantics. Next, we generate a graph with the selected schema items and the words in the question. We connect items and words using attention scores between question tokens and schema item tokens. Since the attention mechanism enables the model to assign relevance between tokens (Galassi et al., 2020), we use the attention scores to predict the schema linking. This allows the model to capture the contextual relationships without any explicit linking dataset. With these, we can effectively achieve a joint representation of each text-to-SQL sample as a graph. By comparing the representations of the graphs, we can retrieve demonstrations. With the demonstrations, our method fully leverages in-context learning capability and generate the target SQL correctly.

To evaluate our proposed method, we conduct experiments on various benchmark datasets, including Spider, Spider-DK, Spider-Realistic, and Spider-Syn (Yu et al., 2018b; Gan et al., 2021b; Deng et al., 2021; Gan et al., 2021a). Moreover, we experiment with various hyper-scaled and small LLMs, including GPT-4, GPT-3.5-Turbo, DeepSeek-Coder-33B, Llama 3.1-8B, DeepSeek-Coder-6.7B, and Llama 3.2-3B (OpenAI et al., 2024; DeepSeek-AI et al., 2024; Dubey et al., 2024). Our method consistently achieves performance improvements not only with hyper-scaled LLMs but also with small LLMs.

2 Related Work

2.1 In-context Learning for Text-to-SQL

With the advancement of large language models, there has been a growing focus on leveraging them to solve Text-to-SQL tasks (Pourreza and Rafiei, 2023; Chang and Fosler-Lussier, 2023; Guo et al., 2024; Zhang et al., 2023a; Lee et al., 2025; Gao et al., 2024; Nan et al., 2023; Li et al., 2024; Shen et al., 2024). Some approaches decompose the Text-to-SQL task into multiple steps and employ GPT-4 at each step (Pourreza and Rafiei, 2023; Li et al., 2024; Lee et al., 2025). However, this repeated use of GPT-4, a hyper-scaled LLM, results in significantly high inference costs. Also, since they heavily depend on the hyper-scaled LLM's capabilities, their performance significantly drops with smaller LLMs, as limited capability of small LLM at each step leads to error accumulation throughout the process.

On the other hand, others (Zhang et al., 2023a; Guo et al., 2024; Nan et al., 2023; Gao et al., 2024; Shen et al., 2024; Li et al., 2024; Lee et al., 2025) tried to leverage in-context learning capability of

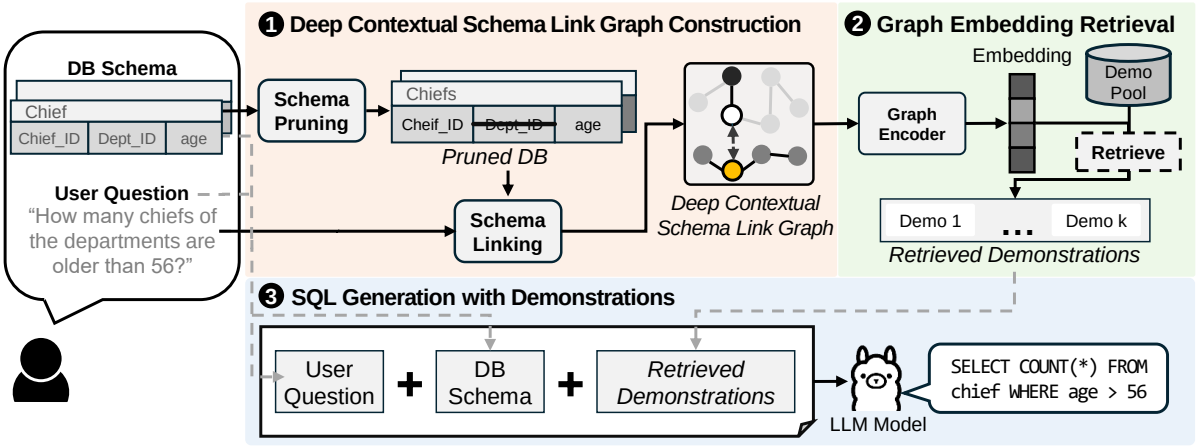

Figure 1: **Overview of our proposed method for Text-to-SQL.** The method consists of three main steps: (1) Deep Contextual Schema Link Graph Construction, which prunes irrelevant schema items and builds a schema link graph that captures the contextual relationship between the question tokens and the schema items; (2) Graph-based Demonstration Retrieval, where graph embeddings are used to retrieve demonstrations relevant to the given query; and (3) SQL Generation using retrieved demonstrations to generate the final SQL query.

LLMs by retrieving relevant samples and providing them as demonstrations. Zhang et al. (2023a); Guo et al. (2024); Li et al. (2024); Lee et al. (2025) encoded the question into a text embedding using a pre-trained text encoder and retrieved demonstrations based on the embedding similarity. However, they did not consider the DB schema, which is also crucial in Text-to-SQL. In contrast, Gao et al. (2024); Nan et al. (2023); Shen et al. (2024) tried to reflect DB schema information indirectly by approximating target SQL of user input and retrieving similar examples through similarity comparison of the approximated SQL. However, they still have the limitation of requiring costly training and relying on the approximator which has limited capacity for generating SQL. As a result, inaccurate DB schema information can be represented when the approximator generates wrong SQL. Since they did not effectively consider the DB schema during retrieval, they failed to provide useful demonstrations for LLMs' in-context learning.

2.2 Graph representation for text-to-SQL samples

Before the advent of LLMs, Text-to-SQL research learned schema linking information, which explicitly associates question tokens with relevant database schema elements, to generate target SQL using encoder-decoder models. This helps the model interpret the question and recognize how schema components are organized within the database, leading to improve SQL generation accuracy (Lei et al., 2020).

To provide schema linking information to the model, Wang et al. (2020); Qi et al. (2022); Zhang et al. (2023b); Li et al. (2023b) represented the input in a graph format, reflecting the relationships between question tokens and database schema elements. Due to the lack of datasets that explicitly annotate schema linking, these methods were based on simple textual matching, comparing only the spelling between question tokens and schema items. However, such simple linking often fails to capture deeper contextual relationship when question token and schema items are expressed in different forms.

Furthermore, their graph representations contain all schema items, even those unrelated to the question, which can lead to incorrect schema linking. Moreover, retrieval becomes challenging when all schema items are included, as each graph is mostly composed of DB schema elements, making samples that share the same DB schema indistinguishable. Therefore, to retrieve useful demonstrations, a more effective graph representation is needed—one that captures deeper contextual relationships while including only the relevant schema elements.

3 Proposed Method

In this section, we introduce a Text-to-SQL method with **D**eep **C**ontextual Schema Link **G**raph-based Retrieval, DCG-SQL. Figure 1 illustrates an overview of our method, consisting of three main processes: Deep Contextual Schema Link Graph Construction, Graph-based Demonstration Retrieval, and SQL Generation.

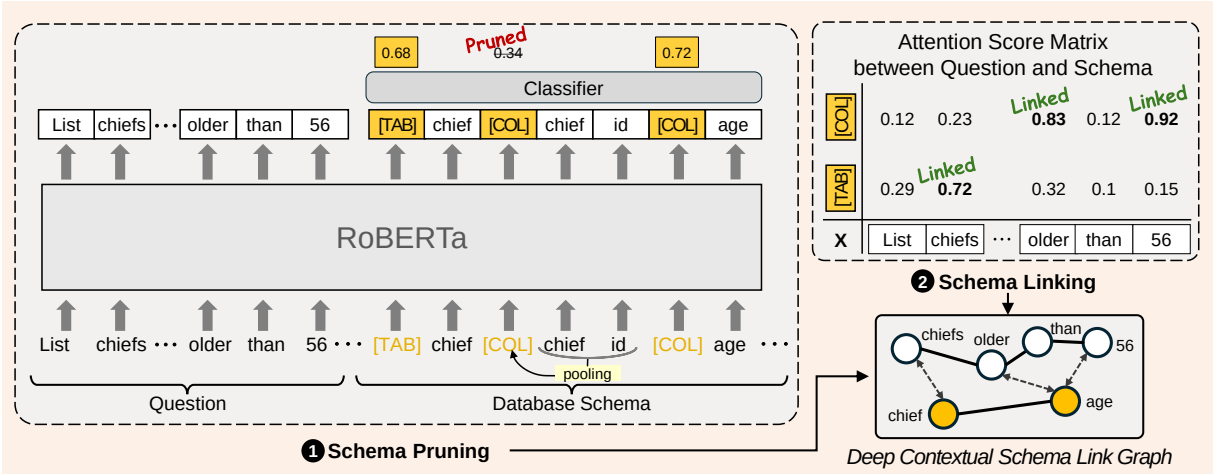

Figure 2: Deep Contextual Schema Link Graph Construction.

First, we construct a deep contextual schema link graph, which captures the contextual relationships between question and relevant schema items. Then, to retrieve demonstrations with our graph representation, we train a graph encoder with self-supervised learning. This ensures that samples closer to the anchor are more useful in generating the target SQL, enabling the effective retrieval of useful demonstrations. Finally, with the selected demonstrations, our method can fully leverage in-context learning ability, and generate the target SQL more correctly.

3.1 Deep Contextual Schema Link Graph Construction

We propose a deep contextual schema link graph construction method that excludes irrelevant schema items and extracts only the relevant ones. Our method also links the question tokens with the schema items based on their contextual relation.

Schema pruning DB schema consists of multiple tables and columns, but only a small subset is relevant for SQL generation. If irrelevant DB schema items are included in the graph, they could hinder extracting representation and retrieving useful demonstrations. To focus on the relevant components, we introduce a schema pruning method that identifies the relevant schema items necessary for generating the SQL.

For schema pruning, we utilize a cross-encoder model to determine the relevance of tables and columns (Li et al., 2023a). The input to the cross-encoder is a flattened text sequence that combines the question with schema items (either a table or a column). Since schema items, such as the column name 'head id', may be tokenized into multiple tokens, they need to be represented as a single token. We prepend special tokens [TAB] and [COL] for representing tables and columns, respectively. The special token embeddings for schema items are initialized by applying mean pooling over the token embeddings of each word in the schema item. Unlike existing pruning approach (Li et al., 2023a), these special tokens enable us to leverage the attention scores between question and schema items.

Let Q denote the tokens of the question, and T and C denote the names of the table and column. The input sequence X is represented as:

$$X = \{Q\,[\text{TAB}]\,T_1\,[\text{COL}]\,C_{11}, \ldots, [\text{COL}]\,C_{1n}\,[\text{TAB}]\,T_2\,[\text{COL}]\,C_{21}, \ldots\}$$

where T_i denotes the i-th table in the DB schema, and C_{ij} refers to the j-th column of the table T_i. The cross-encoder model is trained as binary classification for each special token to predict the relevant tables and columns for a given question, as shown in Figure 2.

Schema linking Identifying schema items contextually relevant to the question is crucial for understanding the question and generating SQL outputs (Lei et al., 2020; Li et al., 2023b; Wang et al., 2020). However, due to the absence of ground truth dataset for such relation, existing approaches rely on simple text correspondence (Li et al., 2023b; Zhang et al., 2023b; Qi et al., 2022; Wang et al., 2020). It compares only the spelling between question tokens and schema items, often failing to capture links when schema items are phrased differently (e.g., linking the question tokens "*older*" and "*56*" to the column "*age*" in Figure 2).

To overcome this issue and capture contextual similarities, we leverage the attention scores from the schema pruning model. Since the pruning model is trained to identify relevant schema items for a given question, its attention scores implicitly reflect the relevance between them. If the schema pruning model classifies a schema item as relevant to SQL generation, we obtain the attention scores between its concatenated special token and the question tokens.

When the attention score exceeds the threshold τ, a link is connected between the schema item and the question token, which we refer to as an 'attention-match'. This approach enables schema pruning and linking simultaneously without ground truth dataset or additional training. Also, we link tokens that have high syntactic correlation for edges between question and all schema items within each table for edges between schema items, as following existing approaches (Qi et al., 2022; Zhang et al., 2023b; Li et al., 2023b; Wang et al., 2020).

Finally, we construct a deep contextual schema linking graph G, where question and schema are jointly represented. The graph consists of nodes, which correspond to question tokens and schema items, and edges that capture the contextual relationships between them.

3.2 Graph-based Demonstration Retrieval

To retrieve useful demonstrations based on the deep contextual schema link graph, we propose a graph-based retrieval method that jointly consider the question and the DB schema. To achieve this, we adopt a graph encoder to effectively represent each sample's graph into an embedding vector (Wang et al., 2020; Qi et al., 2022).

To effectively train the graph encoder, we utilize contrastive learning that clusters similar graphs closely in the vector space while distancing dissimilar ones farther apart. We define positive demonstration samples as those that are useful for generating the SQL query of a given anchor. To achieve this, for the given anchor sample $x = (G_x, SQL_x)$, we extract likelihoods of the SQL_x when other samples $S = \{s_i = (G_i, SQL_i) \mid i = 1, 2, \dots\}$ are provided as demonstrations using Llama 3.1-8B model (Dubey et al., 2024). The likelihood is as follows:

$$score(x, s_i) = P_{\text{sLLM}}(SQL_x \mid G_x; s_i) \quad (1)$$

If the likelihood of SQL_x is high, it indicates that s_i is useful for generating the target SQL query.

To identify positive and negative samples, we rank s_i's based on their likelihood of generating the anchor's target SQL, treating high-ranked ones as positives and low-ranked ones as negatives. Since the retrieval model is trained to increase similarity with high-ranked demonstrations and decrease similarity with low-ranked ones, we can retrieve useful demonstrations from the retrieval pool through similarity of the graph embeddings (Karpukhin et al., 2020; Li et al., 2023c).

However, computing the likelihood of SQL_x over the entire training dataset as demonstrations is computationally expensive. To address this, we construct a reduced candidate set for each anchor by selecting 100 structurally similar SQLs, using a tree edit distance computed over abstract syntax trees (Yu et al., 2018a). Unlike Shen et al. (2024), which selects demonstrations based on AST similarity, we further re-rank the candidates by their usefulness for generating the anchor SQL. This enables us to identify samples that appear structurally similar but are unhelpful as demonstrations—performing hard negative mining.

During inference, we compare the embedding of user input graph G_{user} generated by the encoder, with the embeddings of the demonstration pool by computing the similarity. We select top-k samples with the highest similarity as demonstrations.

3.3 SQL generation with In-context Learning

With the retrieved demonstrations, LLM can generate target SQL query correctly. In addition, we utilized automated-CoT prompting method to guide the SQL generation more effectively (Zhang et al., 2022, 2023a). The retrieved demonstrations are categorized into types such as simple, join, and nested queries. Each type uses a predefined CoT template, dynamically adapted per demonstration to guide consistent and structured SQL generation. Details are provided in Appendix C.2.

4 Experiments

4.1 Implementation Details

All experiments are conducted on an A100 80GB GPU. Our method requires only small size of models for efficiency. For schema link graph construction, we train the schema pruning model based on RoBERTa-large, which has approximately 355 million parameters (Liu et al., 2019). For schema linking, we set τ_{tab} and τ_{col} to 0.66 and 0.43, respectively. For graph-based demonstration re-

Method	Easy		Medium		Hard		Extra		All	
	EX	EM	EX	EM	EX	EM	EX	EM	EX	EM
Finetuned Model										
Graphix-T5	92.3	91.9	86.3	82.3	73.6	65.5	57.2	53.0	80.9	77.1
RESD-SQL	-	-	-	-	-	-	-	-	84.1	80.1
GPT-4										
DIN-SQL	92.3	82.7	87.4	65.5	76.4	42.0	62.7	30.7	82.8	60.1
ACT-SQL	92.3	85.9	89.3	59.1	80.2	53.1	61.5	18.9	83.9	57.9
DAIL-SQL	91.1	90.3	88.6	76.0	75.9	60.3	62.0	50.0	82.8	72.6
ASTRES	-	-	-	-	-	-	-	-	86.6	77.3
DCG-SQL (Ours)	**95.6**	**92.3**	**90.6**	**83.0**	**82.8**	**70.7**	**72.3**	**61.4**	**87.5**	**79.7**
GPT-3.5-Turbo										
DIN-SQL	85.1	56.5	81.8	51.6	62.1	28.2	51.2	7.8	74.4	41.8
ACT-SQL	**92.7**	87.9	86.1	52.5	70.1	41.4	56.6	13.3	80.3	52.8
DAIL-SQL	90.3	**89.9**	82.7	57.0	73.0	52.3	56.0	18.7	78.0	64.9
ASTRES	-	-	-	-	-	-	-	-	83.0	68.8
DCG-SQL (Ours)	91.9	88.3	**89.2**	**78.0**	**74.7**	**56.3**	**63.3**	**39.8**	**83.3**	**70.7**
Deepseek-Coder-33B-Instruct										
DIN-SQL	91.9	81.9	82.1	62.3	58.0	31.0	50.0	17.5	75.0	54.5
ACT-SQL	84.3	78.6	85.0	57.4	64.4	44.8	50.6	25.3	75.8	55.2
DAIL-SQL	90.3	87.9	86.5	73.8	73.0	62.6	57.2	45.2	80.5	70.7
ASTRES	-	-	-	-	-	-	-	-	83.4	64.7
DCG-SQL (Ours)	**94.0**	**90.7**	**92.6**	**80.0**	**75.3**	**63.8**	**63.3**	**53.0**	**85.3**	**75.5**
Llama 3.1-8B-Instruct										
DIN-SQL	84.7	79.4	72.2	54.9	60.9	36.8	35.5	19.9	67.4	52.1
ACT-SQL	89.5	85.9	81.8	63.2	60.9	42.0	50.0	31.9	75.0	60.1
DAIL-SQL	81.9	67.7	75.3	57.4	60.9	37.9	54.8	**46.4**	71.2	54.8
DCG-SQL (Ours)	**94.8**	**93.1**	**87.0**	**78.7**	**77.0**	**57.5**	**55.4**	43.4	**82.1**	**72.9**
Deepseek-Coder-6.7B-Instruct										
DIN-SQL	86.7	70.6	78.7	57.6	59.2	29.3	49.4	21.1	72.6	50.1
ACT-SQL	88.7	82.7	81.4	57.4	66.7	46.6	41.0	20.5	74.2	55.7
DAIL-SQL	85.5	71.0	79.1	63.0	64.9	47.7	48.8	41.6	73.4	58.9
DCG-SQL (Ours)	**91.9**	**89.9**	**86.8**	**78.0**	**74.7**	**56.3**	**56.6**	**48.8**	**81.1**	**72.5**
Llama 3.2-3B-Instruct										
DIN-SQL	69.4	60.1	47.5	35.2	31.6	17.8	14.5	3.6	44.8	33.2
ACT-SQL	85.9	77.8	71.7	56.3	53.4	37.4	33.7	24.1	66.0	53.1
DAIL-SQL	68.1	50.8	63.7	49.6	44.8	25.9	46.4	**42.8**	58.8	44.8
DCG-SQL (Ours)	**91.9**	**87.5**	**79.8**	**71.7**	**60.9**	**48.9**	**50.6**	41.0	**74.7**	**66.7**

Table 2: Experimental Results on Spider Dataset with various LLMs.

trieval, We train a graph encoder model based on a bi-encoder architecture (Karpukhin et al., 2020), where each encoder follows the Relation-Aware Transformer (Shaw et al., 2018) structure initialized with BERT-base (Devlin et al., 2019), totaling approximately 220 million parameters.

For a fair comparison, we reproduce experimental results or follow existing studies. For SQL generation, we use various LLMs, including GPT-4, GPT-3.5-Turbo, DeepSeek-Coder-33B-Instruct, Llama 3.1-8B-Instruct, DeepSeek-Coder-6.7B-Instruct, and Llama 3.2-3B-Instruct (OpenAI et al., 2024; DeepSeek-AI et al., 2024; Dubey et al., 2024). In all experiments, the generation temperature is set to 0. We set 5 demonstrations for the few-shot setting and our input prompt is introduced in Appendix C.1. Training the pruning model took approximately 3 hours, while the graph encoder model was trained for 8 hours. We used spaCy (Honnibal and Montani, 2017) and NLTK (Bird and Loper, 2004) libraries for the edges between question tokens.

4.2 Dataset Details

To verify the effectiveness of our method, we employ the SPIDER dataset (Yu et al., 2018b), large-scale benchmark designed for the text-to-SQL task. Spider spans 200 distinct databases across 138 domains and is widely regarded for its diverse and cross-domain nature. The dataset comprises 7,000 training examples and 1,034 development examples, providing a robust basis for model assess-

Method	DK		Real		Syn	
	EX	EM	EX	EM	EX	EM
GPT-4						
ACT-SQL	72.0	47.7	81.3	55.7	72.1	48.5
DAIL-SQL	-	-	76.0	-	-	-
ASTRES	**72.3**	59.1	80.9	66.1	74.4	61.3
DCG-SQL	71.6	**66.5**	**81.9**	**73.6**	**78.7**	**68.6**
GPT-3.5-Turbo						
DIN-SQL	64.3	36.6	65.2	41.1	65.3	34.7
ACT-SQL	68.2	45.4	76.4	48.6	70.4	44.4
DAIL-SQL	-	-	67.9	-	-	-
ASTRES	68.8	49.3	78.0	60.8	66.9	51.2
DCG-SQL	**69.0**	**59.4**	**79.0**	**62.8**	**74.2**	**61.6**
Deepseek-Coder-33B-Instruct						
ACT-SQL	69.5	47.5	76.4	51.8	69.1	49.3
ASTRES	**70.5**	46.4	77.4	59.3	68.7	49.5
DCG-SQL	68.4	**62.8**	**79.1**	**71.1**	**73.6**	**62.6**
Llama-3.1-8B-Instruct						
DIN-SQL	56.3	44.1	59.6	36.4	57.2	43.5
ACT-SQL	63.6	48.0	74.4	60.2	66.6	53.7
DCG-SQL	**66.5**	**60.2**	**75.8**	**67.1**	**70.1**	**59.7**
Deepseek-Coder-6.7B-Instruct						
DIN-SQL	54.1	16.2	53.5	29.3	53.7	25.4
ACT-SQL	63.2	44.9	69.1	53.5	63.2	44.6
DCG-SQL	**65.0**	**55.5**	**74.8**	**68.5**	**70.5**	**59.9**
Llama-3.2-3B-Instruct						
DIN-SQL	35.7	27.5	38.6	29.1	39.0	27.2
ACT-SQL	54.4	41.7	62.0	47.0	57.4	44.4
DCG-SQL	**59.1**	**52.5**	**68.5**	**59.6**	**60.9**	**51.9**

Table 3: Experimental Results on Spider-DK, Spider-Realistic, and Spider-Syn with various LLMs.

ment. The dataset categorizes SQL queries based on their complexity into four difficulty levels: easy, medium, hard, extra hard. Our evaluation experiments are conducted on the development set, using the training set as the demonstration pool for retrieval. Moreover, using the same demonstration pool, we evaluate on Spider-DK (Gan et al., 2021b), Spider-Realistic (Deng et al., 2021), and Spider-Syn (Gan et al., 2021a), which are derived from Spider. They incorporate domain knowledge, remove explicitly mentioned column names, and replace schema-related words with synonyms, respectively.

4.3 Baselines & Evaluation Metrics

We compared our method with recent state-of-the-art in-context learning baselines for Text-to-SQL (Pourreza and Rafiei, 2023; Zhang et al., 2023a; Gao et al., 2024; Shen et al., 2024).

We use two evaluation metrics, Execution Accuracy (EX) and Exact-set-match Accuracy (EM). Execution accuracy measures whether the predicted SQL query yields the same result as the ground truth query when executed. Exact-set-match accuracy treats each clause as a set and compares the predicted clause to the ground truth clause.

Retrieval	Spider		Real		Syn	
	EX	EM	EX	EM	EX	EM
GPT-3.5-Turbo						
Random	78.8	47.0	74.6	42.2	60.0	38.4
DCG-SQL	**83.3**	**70.7**	**79.0**	**62.8**	**74.2**	**61.6**
Llama-3.1-8B-Instruct						
Random	73.0	48.5	69.7	45.5	61.6	39.7
DCG-SQL	**82.1**	**72.9**	**75.8**	**67.1**	**70.1**	**59.7**

Table 4: Effectiveness of our retrieved demonstrations.

4.4 Experimental Results

Table 2 presents the Text-to-SQL performance on the Spider dataset using various hyper-scaled LLMs and small LLMs. Our proposed method consistently achieves higher execution accuracy and exact match accuracy compared to existing approaches. In particular, our method achieves notable improvements for sLLMs. For execution accuracy (EX), it increases by 9.5% with Llama 3.1-8B and 9.3% with DeepSeek-Coder-6.7B. Notably, there is a substantial gap in exact match accuracy (EM), increasing by 21.3% with Llama 3.1-8B and 23.1% with DeepSeek-Coder-6.7B. This suggests that our demonstrations provide the precise patterns needed for the test cases, as in-context learning relies on pattern following (Brown et al., 2020; Luo et al., 2024). These results indicate that our approach, which leverages deep contextual schema link graph-based retrieval for SQL generation, effectively utilizes the in-context learning capabilities, leading to outperforming existing methods in text-to-SQL task.

Additionally, to demonstrate the effectiveness of our approach in more challenging scenarios, we conduct experiments on Spider-DK, Spider-Realistic, and Spider-Syn for SQL generation, as shown in Table 3. Our method consistently demonstrates strong performance across various LLM scenarios and datasets. These findings highlight the effectiveness of our method across different datasets, showing strong performance not only with hyper-scaled LLMs but also with small LLMs.

Retrieval Case	EX (%)	EM (%)
(1)	77.0	66.2
(2)	77.1	65.6
(3)	76.2	66.5
(4)	76.1	59.6
Ours	**82.1**	**72.9**

Table 5: Effectiveness of our deep contextual schema link graph-based retrieval. The experiments were conducted on Spider with Llama-3.1-8B-Instruct.

DB Schema Type	Precision	Recall	F1 Score
Table	96.2	97.4	96.8
Column	90.7	95.5	93.1

Table 6: Schema pruning performance.

4.5 Ablation Studies

Effectiveness of our retrieval method. Table 4 presents the results when demonstrations are randomly chosen. Existing approaches (Zhang et al., 2023a; Gao et al., 2024) failed to maximize in-context learning capabilities due to their ineffective retrieval approaches as shown in Table 1, whereas our method consistently achieves performance improvements across various datasets and LLMs.

To demonstrate the effectiveness of our retrieval method, we compare it with various retrieval approaches, as shown in Table 5. For demonstration retrieval, case (1) is based on the text embedding of question and (2) is based on the masked question text embedding (Guo et al., 2024). Case (3) is based on both question and DB schema, but uses text embeddings. In case (4), it is based on the existing graph representation (Li et al., 2023b). Cases (1)–(3) use BERT-base as the text encoder, whereas case (4) employs the same graph encoder as our method. All (1)–(4) are trained using the same self-supervised learning approach described in Section 3.2

As shown in cases (1) and (2), demonstration retrieval based only on question does not achieve good performance. Also, retrieval methods that consider both question and DB schema information without their contextual relationships, show poor performance. In contrast, our method captures contextual relationships in the joint representation of Text-to-SQL samples, leading to better demonstration retrieval.

Method	EX (%)	EM (%)
DAIL-SQL	71.2	54.8
DAIL-SQL w/ Our Retrieval	**75.6**	**61.3**

Table 7: Transferability of our proposed modules. The experiments were conducted on Spider with Llama-8B.

Method	Average Tree Edit Distance
ACT-SQL	27.17
DAIL-SQL	22.38
DCG-SQL	**13.82**

Table 8: Average Tree Edit Distance between demonstration and target SQL query on Spider Dataset

Effectiveness of our DB schema pruning. We demonstrate the effectiveness of our pruning method as shown in Table 6. For tables, our method achieves a precision of 96.2%, recall of 97.4%, and an F1 score of 96.8%. For columns, our method shows a precision of 90.7%, recall of 95.5%, and an F1 score of 93.1%. These results show that our method effectively identifies relevant schema items and exclude irrelevant links.

Transferability of our proposed modules. To demonstrate the transferability of our modules, we applied our retrieval to the DAIL-SQL (Gao et al., 2024). All the experimental results in Table 7 are obtained using the same setting as DAIL-SQL. When we use the demonstrations sets retrieved by our method, the performance is 75.6% in EX and 61.3% in EM, respectively. This corresponds to improvements of 6.2% and 11.9% over the original DAIL-SQL. These results demonstrate that our retrieval method can be effectively applied to various prompt templates and Text-to-SQL methods.

Analysis of retrieved demonstrations. To analyze the quality of retrieved demonstrations, we calculate the average Tree Edit Distance (TED) (Yu et al., 2018a), which measures the similarity between the target SQL of user input and the SQLs of the retrieved demonstrations. If TED is small, it indicates that the SQLs of retrieved demonstrations are similar to the target SQL of user input. As shown in the Table 8, our method achieves the lowest TED compared to existing approaches. Since in-context learning of LLMs tends to follow the patterns in the demonstrations (Brown et al., 2020; Luo et al., 2024), the result supports that our proposed method effectively retrieves useful demonstrations for generating correct SQL.

Method	EX (%)	Retrieval Params.	LLM Calls	Retrieval Latency
DIN-SQL	67.4	-	4	-
ACT-SQL	75.0	1M	1	0.7
DAIL-SQL	71.2	3B	1	9.1
DCG-SQL	82.1	0.5B	1	1.3

Table 9: Comparison of Efficiency. Experimental results are measured on Spider with Llama-3.1-8B-Instruct.

Efficiency of our retrieval method. To demonstrate the efficiency of our method, we evaluate the inference latency on the Spider dataset, as shown in Table 9. Retrieval latency refers to the time required on average to extract embeddings from test cases. DIN-SQL (Pourreza and Rafiei, 2023), which performs four separate LLM inferences without retrieval, incurs four times the LLM latency compared to other methods. Meanwhile, DAIL-SQL (Gao et al., 2024) employs a retrieval model with 3B parameters (Li et al., 2023b), resulting in latency up to 9.1 seconds. Although ACT-SQL (Zhang et al., 2023a) shows relatively low inference latency, its execution accuracy remains unsatisfactory. In contrast, our method requires only a single LLM inference and employs lightweight retrieval modules with just 0.5B trainable parameters, including the schema pruning model and the graph encoder. This design enables more efficient retrieval while maintaining high execution accuracy. These results clearly demonstrate the effectiveness and efficiency of our approach.

5 Conclusion

This paper presents a novel approach for retrieving demonstrations and generating SQL queries, effective across LLMs of various sizes. Our method leverages a deep contextual schema link graph to enhance demonstration retrieval and SQL generation. It achieved strong performance across hyper-scaled and small LLMs.

Limitations

Under the same LLM models, our method outperforms the existing SOTA. Our method has shown highly effective in improving performance with small LLMs, however, there is still a performance gap between the use of hyper-scaled LLMs and small LLMs. Future work should focus on addressing this performance gap effectively.

Acknowledgments

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.IITP-2025-RS-2025-00437633, Information Technology Research Center, 20%), (No.RS-2019-II190421, AI Graduate School Support Program(Sungkyunkwan University), 20%), (No.RS-2025-02218768, Accelerated Insight Reasoning via Continual Learning, 20%). This work was also supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.RS-2025-00521391, 20%) and Artificial intelligence industrial convergence cluster development project funded by the Ministry of Science and ICT(MSIT, Korea)&Gwangju Metropolitan City (20%).

References

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

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

Shuaichen Chang and Eric Fosler-Lussier. 2023. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. *arXiv preprint arXiv:2305.11853*.

DeepSeek-AI, :, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *Preprint*, arXiv:2401.02954.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for*

Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Andrea Galassi, Marco Lippi, and Paolo Torroni. 2020. Attention in natural language processing. *IEEE transactions on neural networks and learning systems*, 32(10):4291–4308.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. 2024. Prompting gpt-3.5 for text-to-sql with desemanticization and skeleton retrieval. In *PRICAI 2023: Trends in Artificial Intelligence*, pages 262–274, Singapore. Springer Nature Singapore.

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 *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, Online. Association for Computational Linguistics.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13076–13084.

Xiaonan Li, Kai Lv, Hang Yan, Tianyang Lin, Wei Zhu, Yuan Ni, Guotong Xie, Xiaoling Wang, and Xipeng Qiu. 2023c. Unified demonstration retriever for in-context learning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4644–4668, Toronto, Canada. Association for Computational Linguistics.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency. *Preprint*, arXiv:2403.09732.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Man Luo, Xin Xu, Yue Liu, Panupong Pasupat, and Mehran Kazemi. 2024. In-context learning with retrieved demonstrations for language models: A survey. *arXiv preprint arXiv:2401.11624*.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing text-to-SQL capabilities of large language models: A study on prompt design strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14935–14956, Singapore. Association for Computational Linguistics.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems*, volume 36, pages 36339–36348. Curran Associates, Inc.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. RASAT: Integrating relational structures into pretrained Seq2Seq model for text-to-SQL. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3215–3229, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *Preprint*, arXiv:2204.00498.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, Kaustubh Vyas, Yuanyi Ji, and Jeff Z. Pan. 2024. Improving retrieval-augmented text-to-SQL with AST-based ranking and schema pruning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7865–7879, Miami, Florida, USA. Association for Computational Linguistics.

Yuan Tian, Zheng Zhang, Zheng Ning, Toby Jia-Jun Li, Jonathan K. Kummerfeld, and Tianyi Zhang. 2023. Interactive text-to-SQL generation via editable step-by-step explanations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 16149–16166, Singapore. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models. *Preprint*, arXiv:2206.07682.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

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. 2018b. 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. Association for Computational Linguistics.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023a. ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3501–3532, Singapore. Association for Computational Linguistics.

Kun Zhang, Xiexiong Lin, Yuanzhuo Wang, Xin Zhang, Fei Sun, Cen Jianhe, Hexiang Tan, Xuhui Jiang, and Huawei Shen. 2023b. ReFSQL: A retrieval-augmentation framework for text-to-SQL generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 664–673, Singapore. Association for Computational Linguistics.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *Preprint*, arXiv:2210.03493.

A Generalization Performance on BIRD dataset

To verify the generalization ability of our method, we conduct experiments on the BIRD dataset using the same hyperparameters and evaluate its performance on the BIRD dev dataset. ACT-SQL (Zhang et al., 2023a) cannot be evaluated on the BIRD dataset because it relies heavily on the Spider dataset. In addition, DIN-SQL (Pourreza and Rafiei, 2023) cannot be evaluated with Llama 3.1-8B-Instruct model because input prompts, provided in their official code, exceed the maximum token length of the model which is 8,192. As shown in the Table 10, existing methods exhibit similar performance to random demonstrations, while our method outperforms them, indicating its strong generalization performance.

Method	EA	VES
GPT-3.5-Turbo		
DIN-SQL	45.1	52.4
DAIL-SQL	45.9	50.6
DCG-SQL	**49.7**	**54.4**
Deepseek-Coder-33B-Instruct		
DIN-SQL	48.8	53.4
DAIL-SQL	49.9	52.6
DCG-SQL	**53.2**	**56.0**
Llama 3.1-8B-Instruct		
DAIL-SQL	38.40	39.38
DCG-SQL	**42.20**	**46.67**
Deepseek-Coder-6.7B-Instruct		
DIN-SQL	41.7	44.8
DAIL-SQL	40.7	43.8
DCG-SQL	**44.8**	**47.2**

Table 10: Performance on BIRD dataset with various LLMs. EA and VES indicate Execution Accuracy and Valid Efficiency Score, respectively.

B Analysis of Qualitative Results

To ensure a reliable qualitative analysis, we compare the generated SQL queries with those from existing approaches (Gao et al., 2024; Zhang et al., 2023a) by using diverse cases of questions and DB schemas. Figure 3 and Figure 4 demonstrate the question, database, and selected demonstrations, which are used for SQL query generation. In the existing approaches, the given database contains all schema items, making the model choose irrelevant schema items for SQL query generation. Additionally, the selected demonstrations differ significantly in syntax from the target SQL query. Such difference cause the model to generate incorrect SQL queries since in-context learning makes LLMs follow the patterns in the examples (Brown et al., 2020). In contrast, our method effectively includes relevant schema items and the selected demonstrations contain the highly similar to target SQL query. These indicate that our method successfully identifies the useful DB schema information and demonstrations for generating the target SQL.

C Prompt Details

In this section, we presents all the prompts of our methods. Our prompt is based on API DOCS format (Rajkumar et al., 2022). Following (Lin et al., 2020), we also provide potentially useful values by performing if there is match between question tokens and the values of each column.

C.1 Test Case Prompt

We demonstrate the prompt for a test case in Table 11.

C.2 Automated-CoT prompt

As shown in Table 12, we categorized the training samples based on the SQL into four cases, SIMPLE, JOIN, NESTED, and IUEN. We designed automated-CoT prompt beginning with "Let's think step by step." except for SIMPLE cases. For JOIN, we make the model to generate intermediate step first, then generate final query (Pourreza and Rafiei, 2023). For nested and IEUN(INTERSECT, EXCEPT, UNION, NOT IN), we added a natural language phrase to describe briefly about the nested SQL query by converting WHERE clause and HAVING clause of the nested query with rule-based approach (Tian et al., 2023).

	ACT-SQL	Ours
Example 1		
Question	Find the name of the shops that do not hire any employee.	
Given Database Schema	**employee** (employee_id , name , age , city) **shop** (shop_id , name , ⋯ , manager_name) **hiring** (shop_id , employee_id , ⋯ , is_full_time) **evaluation** (employee_id , year_awarded , bonus) Foreign Keys = [hiring.shop_id = shop.shop_id , ⋯ , evaluation.employee_id = employee.employee_id]	
Selected Demonstration	SELECT name FROM products	SELECT name FROM driver WHERE driver_id NOT IN (SELECT driver_id FROM school_bus)
Generated SQL	SELECT T2.name FROM employee AS T1 JOIN hiring AS T2 ON T1.employee_id = T2.employee_id GROUP BY T2.shop_id HAVING COUNT(*) = 0 ✗	SELECT name FROM shop WHERE shop_id NOT IN (SELECT shop_id FROM hiring) ✓
Example 2		
Question	Find the number of professionals who have ever treated dogs.	
Given Database Schema	**sizes** (size_code , size_description) **owners** (owner_id , first_name , ⋯ , cell_number) **dogs** (dog_id, owner_id, ⋯ , date_of_birth) professionals (professional_id , ⋯ , cell_number) **treatments** (dog_id , professional_id , ⋯) Foreign Keys = [dogs.owner_id = owners.owner_id, dogs.owner_id = owners.owner_id , ⋯]	
Selected Demonstration	SELECT COUNT(*) FROM enzyme WHERE id NOT IN (SELECT enzyme_id FROM medicine_enzyme_interaction)	SELECT COUNT(DISTINCT rank) FROM captain
Generated SQL	SELECT COUNT(*) FROM professionals WHERE professional_id IN (SELECT professional_id FROM treatments) ✗	SELECT COUNT(DISTINCT professional_id) FROM treatments ✓
Example 3		
Question	When was the last transcript released?	
Given Database Schema	**addresses** (address_id , line_1, ⋯ , country) **departments** (department_id , ⋯ , other_details) **sections** (section_id , course_id , ⋯ , other_details) **transcript_contents** (student_course_id , ⋯ , transcript_id) **transcripts** (transcript_id , transcript_date , ⋯) Foreign Keys = [transcript_contents.transcript_id = transcripts.transcript_id , ⋯]	
Selected Demonstration	SELECT COUNT(*) FROM list AS T1 JOIN teachers AS T2 ON T1.classroom = T2.classroom WHERE T1.firstname = “MADLOCK” AND T1.lastname = “RAY”	SELECT official_name , status FROM city ORDER BY population DESC LIMIT 1
Generated SQL	SELECT T1.transcript_data FROM transcripts AS T1 JOIN transcript_contents AS T2 ON T1.transcript_id = T2.transcript_id WHERE T2.student_course_id = 76 ✗	SELECT transcript_date FROM transcripts ORDER BY transcript_date DESC LIMIT 1 ✓

Figure 3: Qualitative results. These results demonstrate a comparison between our generated queries and those from ACT-SQL.

	DAIL-SQL	Ours
Example 1		
Question	What is the name and nation of the singer who have a song having “Hey” in its name?	
Given Database Schema	**stadium** (stadium_id , location , ⋯ , average) **singer** (name , country , ⋯ , song_name , is_male) **concert** (concert_id , concert_name , ⋯ , year) **singer_in_concert** (concert_id , singer_id) Foreign Keys = [singer_in_concert.singer_id = singer.singer_id , ⋯ , singer_in_concert.concert_id = concert.concert_id]	
Selected Demonstration	SELECT T2.name FROM assignedto AS T1 JOIN projects AS T2 ON T1.project = T2.code JOIN scientists AS T3 ON T1.scientist = T3.SSN WHERE T3.name LIKE ‘%Smith%’	SELECT cust_name , acc_bal FROM customer WHERE cust_name LIKE ‘%a%’
Generated SQL	SELECT T1.name , T1.country FROM singer AS T1 JOIN singer_in_concert AS T2 ON T1.singer_id = T2.singer_id JOIN concert AS T3 ON T2.concert_id = T3.concert_id WHERE T1.name LIKE ‘%HEY%’ ✗	SELECT name , country FROM singer WHERE song_name LIKE ‘%HEY%’ ✓
Example 2		
Question	Find the name of tourney that has more than 10 matches	
Given Database Schema	**players** (player_id , first_name , ⋯ , country_code) **matches** (draw_size , tourney_name , ⋯ , year) **rankings** (ranking_data , ranking , player_id , ⋯ , tours) Foreign Keys = [matches.winner_id = players.player_id , matches.loser_id = players.player_id , rankings.player_id = players.player_id]	
Selected Demonstration	SELECT T2.first_name, T2.last_name FROM customer_payments AS T1 JOIN customers AS T2 ON T1.customer_id = T2.customer_id GROUP BY T1.customer_id HAVING COUNT(*) > 2	SELECT class FROM captain GROUP BY class HAVING COUNT(*) > 2
Generated SQL	SELECT T1.tourney_name FROM matches AS T1 JOIN tournaments AS T2 ON T1.tourney_id = T2.tourney_id GROUP BY T1.tourney_id HAVING COUNT(*) > 10 ✗	SELECT tourney_name FROM matches GROUP BY tourney_name HAVING COUNT(*) > 10 ✓
Example 3		
Question	What are the airline names and abbreviations for airlines in the USA?	
Given Database Schema	**airlines** (uid , airline , abbreviation , country) **airports** (city , airportcode , ⋯ , countryabbrev) **flights** (airline , flightno , sourceairport , destairport) Foreign Keys = [flights.destairport = airports.airportcode , flights.sourceairport = airports.airportcode]	
Selected Demonstration	SELECT T1.employee_id , T2.job_title FROM employees AS T1 JOIN jobs AS T2 ON T1.job_id = T2.job_id WHERE T1.department_id = 80	SELECT open_date , open_year FROM shop WHERE shop_name = ‘Apple’
Generated SQL	SELECT T1.airline , T1.abbreviation FROM airlines AS T1 JOIN airports AS T2 ON T1.country = T2.country WHERE T2.country = ‘USA’ ✗	SELECT airline , abbreviation FROM airlines WHERE country = ‘USA’ ✓

Figure 4: Qualitative results. These results demonstrate a comparison between our generated queries and those from DAIL-SQL.

Prompt
Answer the question by SQLite SQL query only and with no explanation. ### SQLite SQL tables, with their properties: # # city (id, name, countrycode, district, population) # sqlite_sequence (name,seq) # country (code, name, continent, region, surfacearea, indepyear, population, lifeexpectancy, gnp, gnpold, localname, governmentform, headofstate, capital, code2) # countrylanguage (countrycode, language [Dutch, English], isofficial, percentage) # Foreign Keys = [city.countrycode = country.code, countrylanguage.countrycode = country.code] # ### Question: Which regions speak Dutch or English?

Table 11: Test Case Prompt Examples.

	Prompt
Simple	### Answer the question by SQLite SQL query only and with no explanation. ### SQLite SQL tables, with their properties: # # head (age,...) # Foreign Keys = [...] # ### Question: How many heads of the departments are older than 56 ? ### SQL: SELECT COUNT(*) FROM head WHERE age > 56 ;
JOIN	### Answer the question by SQLite SQL query only and with no explanation. ### SQLite SQL tables, with their properties: # # station (id, name, long, ...) # trip (duration, start_station_id, ...) # Foreign Keys = [...] # ### Question: For each station, return its longitude and the average duration of trips that started from the station. Let‘s think step by step. we need to join the tables ’station’ and ’trip’. Create an intermediate representation, then use it to construct the query. Intermediate representation: "FROM station AS T1 JOIN trip AS T2 ON T1.id = T2.start_station_id". ### SQL: SELECT T1.name , T1.long , AVG(T2.duration) FROM station AS T1 JOIN trip AS T2 ON T1.id = T2.start_station_id GROUP BY T2.start_station_id ;
NESTED	### Answer the question by SQLite SQL query only and with no explanation. ### SQLite SQL tables, with their properties: # # weather (date, min_dew_point_f, zip_code, ...) # Foreign Keys = [...] # ### Question: On which day and in which zip code was the min dew point lower than any day in zip code 94107? Let‘s think step by step. we need a nested subquery for ’zip code of the weather is 94107’. Nested subquery: "(SELECT min (min_dew_point_f) FROM weather WHERE zip_code = 94107)". With the nested subquery, we can get the final SQL query. ### SQL: SELECT date , zip_code FROM weather WHERE min_dew_point_f < (SELECT MIN(min_dew_point_f) FROM weather WHERE zip_code = 94107) ;
IUEN	### Answer the question by SQLite SQL query only and with no explanation. ### SQLite SQL tables, with their properties: # # city (status, population,...) # Foreign Keys = [...] # ### Question: Show the status shared by cities with population bigger than 1500 and smaller than 500. Let‘s think step by step. The question can be solved using the ’INTERSECT’ set operator and two subqueries: one for ’population of the city is above 1500’ and the other for ’population of the city is less than 500’. First subquery: SELECT Status FROM city WHERE Population > 1500 Second subquery: SELECT Status FROM city WHERE Population < 500 With the nested subquery, we can get the final SQL query. ### SQL: SELECT status FROM city WHERE population > 1500 INTERSECT SELECT status FROM city WHERE population < 500 ;

Table 12: Automated-CoT for demonstrations.