Retrieval-Augmented GPT-3.5-Based Text-to-SQL Framework with Sample-Aware Prompting and Dynamic Revision Chain

Chunxi Guo, Zhiliang Tian[✉], Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang[✉]

College of Computer, National University of Defense Technology, Changsha, China
{chunxi,tianzhiliang,tangjintao,shashali,zhwen,wangkaixuan18, tingwang}@nudt.edu.cn

Abstract. Text-to-SQL aims at generating SQL queries for the given natural language questions and thus helping users to query databases. Prompt learning with large language models (LLMs) has emerged as a recent approach, which designs prompts to lead LLMs to understand the input question and generate the corresponding SQL. However, it faces challenges with strict SQL syntax requirements. Existing work prompts the LLMs with a list of demonstration examples (i.e. question-SQL pairs) to generate SQL, but the fixed prompts can hardly handle the scenario where the semantic gap between the retrieved demonstration and the input question is large. In this paper, we propose a retrieval-augmented prompting method for an LLM-based Text-to-SQL framework, involving sample-aware prompting and a dynamic revision chain. Our approach incorporates sample-aware demonstrations, which include the composition of SQL operators and fine-grained information related to the given question. To retrieve questions sharing similar intents with input questions, we propose two strategies for assisting retrieval. Firstly, we leverage LLMs to simplify the original questions, unifying the syntax and thereby clarifying the users' intentions. To generate executable and accurate SQLs without human intervention, we design a dynamic revision chain that iteratively adapts fine-grained feedback from the previously generated SQL. Experimental results on three Text-to-SQL benchmarks demonstrate the superiority of our method over strong baseline models.

Keywords: Large language model · Text-to-SQL · Prompt learning

1 Introduction

Text-to-SQL task aims to convert natural language question (NLQ) to structured query language (SQL), allowing non-expert users to obtain desired information from databases [1,2]. As databases are popular in various scenarios involving different domains (e.g., education and financial systems, etc.), it is desirable to train

a model that generalizes well across multiple domains. To facilitate cross-domain generalization [3,4], researchers adapt encoder-decoder architecture [5,6], reducing the requirement for specific domain knowledge via end-to-end training. These approaches require diverse and extensive training data to train the model, which is prohibitively expensive [7].

Recent progress focuses on large language models (LLMs) (e.g., GPT-3 [8], Codex [9] and GPT-4 [10]) with prompt learning [11], which refers to using specific prompts or instructions to generate desired responses. Rajkumar et al. [12] and Liu et al. [13] evaluate several prompt learning baselines for Text-to-SQL tasks. Their findings show that though it is natural for LLMs to generate text sequences, generating SQL is still a challenge due to the SQL's strict syntax requirements. To address these issues, inspired by few-shot learning [11], existing work employs prompting the LLMs with a list of demonstration examples (i.e. question-SQL pairs) to generate SQL queries. However, they typically rely on manual labour to create static demonstration examples tailored to specific tasks. DIN-SQL [14] selects pre-defined samples from each category, SELF-DEBUGGING [15] explains the code to LLMs but without explanation demonstration. These methods employ a static demonstration, meaning that the demonstration examples provided to LLMs are fixed and do not adapt or change across different examples. These static demonstration examples hardly adapt to the scenarios where the semantic gap between retrieved demonstrations and the input question is large, which is called retrieval bias [16], commonly appearing in the retrieval-augmented generation.

Inspired by [17], we argue that providing dynamic demonstrations can be adaptive to specific samples and schema for SQL generation. Dynamic examples enable the SQL generation to accommodate various scenarios. By adjusting to specific instances, demonstrations can be customized to incorporate the necessary query structure, logical operations, and question semantics. This adaptability makes it easier to generate SQL queries that are relevant and appropriate for various situations.

In this paper, we propose retrieval-augmented prompts for an LLM-based Text-to-SQL model, which contains sample-aware prompting and a dynamic revision chain. Specifically, we propose to retrieve similar SQL queries to construct prompts with sample-aware demonstration examples. Notice that users often ask questions in different expressions, even if they have the same intention and SQL query. It makes the model hard to retrieve helpful examples. To solve this issue, we propose to extract the question's real intention via two strategies: Firstly, we simplify original questions through LLMs to clarify the user's intentions and unify the syntax for retrieval. Secondly, we extract question skeletons for retrieving items with similar question intents. To produce executable and accurate SQL, we design a dynamic revision chain, generating SQL queries by iteratively adapting to fine-grained feedback according to the previous version of the generated SQL. The feedback includes SQL execution results, SQL explanations, and related database contents. This dynamic chain manages to generate

executable and accurate SQL through automatic interaction between the language model and the database without human intervention.

Our contributions are as follows: (1) We develop a retrieval-augmented framework for Text-to-SQL tasks by prompting LLMs with sample-aware demonstrations. (2) We propose a dynamic revision chain, which adapts to the previously generated SQL with fine-grained feedback. (3) Our method outperforms strong baseline models on three Text-to-SQL benchmarks.

2   Related Work

2.1   Encoder-Decoder SQL Generation

SQL generation tasks have achieved significant advancements through the utilization of encoder-decoder architectures [2].

On the encoder side, Guo et al. [18] proposed IRNET, using attention-based Bi-LSTM for encoding and an intermediate representation-based decoder for SQL prediction. Later, [19,20] introduced graph-based encoders to construct schema graphs and improve input representations. Works such as RATSQL [1], SDSQL [5], LGESQL [21], S^2SQL [22], R^2SQL [23], SCORE [24], and STAR [25] further improved structural reasoning by modelling relations between schemas and questions. GRAPHIX-T5 [6] overcomes the limitations of previous methods by incorporating graph representation learning in the encoder. Concurrently, RASAT [26] also provided T5 with structural information by adding edge embedding into multi-head self-attention.

On the decoder side, we divide the methods into four categories: sequence-based methods (BRIDGE [27], PICARD [3]) directly translate NLQ into SQL query token by token, template-based methods (X-SQL [28], HydraNet [29]) employ predefined templates to regulate SQL generation and ensure structural coherence, stage-based methods (GAZP [30], RYANSQL [31]) first establish a coarse-grained SQL framework and then fills in the missing details in the frame which calls slot-filling methodologies, and hierarchical-based methods (IRNet [32], RAT-SQL [1]) generate SQL according to grammar rules in a top-down manner, resulting in a tree-like structure.

2.2   LLM-Based SQL Generation

LLM-based models recently emerge as a viable option for this task [12,13]. For effectively utilizing, it is important to design appropriate in-context demonstration [33] and chain-of-thought (CoT) [34] strategies that can elicit its ability [7].

In terms of searching for demonstrations, DIN [14] selects a fair number of demonstration examples from each category (e.g. simple classes, non-nested complex classes and nested complex classes), but they are fixed. Moreover, Guo et al. [35] adaptively retrieve intention-similar SQL demonstration examples through de-semanticization of the questions. However, none of these methods can solve the ambiguous and varied questioning of realistic scenarios.
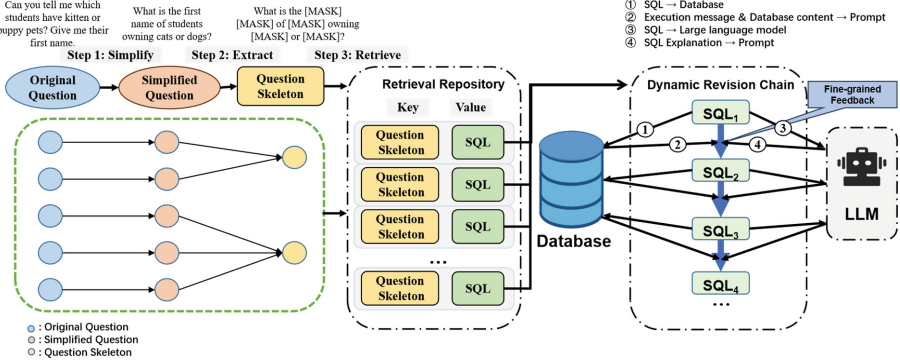
Fig. 1. Framework overview: The left half shows retrieval repository construction in three steps. The top three sentences are three specific instances each. The Green dashed box presents the training set. The right half is a dynamic revision chain with SQL queries generated by LLM iterations as nodes (green boxes). The output of steps 2 and 4 are collectively referred to as fine-grained feedback. (Color figure online)

As for the CoT prompting strategy, DIN-SQL [14] follows a least-to-most [36] prompting method, decomposing Text-to-SQL task into subtasks and solves them one by one. Pourreza and Chen et al. explore self-correction [14,15], where the LLM explain the question and SQL, providing valuable feedback for improvement. Tian et al. [37] propose interactive generation with editable step-by-step explanations, combining human intervention with LLM generation to refine the final SQL output. Additionally, Sun et al. [38] explore execution-based self-consistent prompting methods.

Nonetheless, creating task-specific demonstration examples [14,15,37,38] demands manual labour. Inspired by some retrieval-related research [16,39,40], we develop a retrieval-augmented framework for Text-to-SQL tasks. Our method works through automatic interaction between the LLMs and the databases without human intervention. Moreover, explaining to itself and simple feedback alone [14,15] are weak for digging out errors for correction. Our approach takes into account all three aspects of fine-grained feedback, which interact with each other to create effective feedback.

3   Methodology

Our framework consists of two modules as shown in Fig. 1: (1) **Retrieval Repository:** (see Sect. 3.1) We construct a retrieval repository with simplified questions added and then use question skeletons to retrieve sample-aware SQL demonstration examples. (2) **Dynamic Revision Chain:** (see Sect. 3.2) We further revise the generated SQL queries by adding fine-grained feedback.

3.1 Retrieval Repository

We construct a retrieval repository consisting of multiple key-value retrieval items, where the keys represent the question skeletons and the values are k sample-aware SQL queries. These processes enable us to generate demonstration examples that showcase the desired behaviours of the LLM. Our method involves: (1) Simplifying original questions to unify various questioning styles (see Sect. 3.1.1). (2) Extracting question skeletons to construct a retrieval repository (see Sect. 3.1.2). (3) Retrieving SQL queries according to skeleton similarities (see Sect. 3.1.3).

3.1.1 Question Simplification

We simplify natural language questions by prompting the LLM with instructions. In this way, we can avoid the frustration of unusual questioning styles and enhance the syntax and wording variety in the repository.

Specifically, we construct a prompt template *prompt(.)*: *"Replace the words as far as possible to simplify the question, making it syntactically clear, common and easy to understand: [QUESTION]"*, where *"[QUESTION]"* represents the original natural language question. We then obtain the simplified question by feeding $prompt(Q)$ into the LLM. We maintain a consistent temperature setting in the language model to ensure that all simplified sentences exhibit the same probability distribution.

3.1.2 Question Skeleton Extraction

We then extract question skeletons, including both original questions and simplified questions. We follow the method proposed by Guo et al. [35] to obtain question skeletons. This process removes specific schema-related tokens from the questions, focusing solely on the structure and intent. Finally, we take the (question skeleton, SQL) pairs from the training set and store them in the retrieval repository. Note that the number of samples in the retrieval repository is twice as large as the training set, due to the addition of the simplified samples.

Let $\mathcal{D}_{\text{train}}$ represent the training set, and R denotes the retrieval repository. The original natural language question is denoted as Q_{o}, while Q_{r} represents the simplified question. The question skeletons are denoted as S_{o} and S_{r} for the original and simplified questions, respectively. We formalize the composition of the retrieval repository as follows:

$$R = \{(S_{\text{o}}, \text{SQL}), (S_{\text{r}}, \text{SQL}) \mid (Q_{\text{o}}, \text{SQL}) \in \mathcal{D}_{\text{train}}\}.$$

3.1.3 Sample Retrieval

The retrieval process searches for the most similar question skeletons and returns their corresponding SQL queries from the retrieval repository. This search is based on the semantic similarity between the skeleton of the new question and the items' keys in R.

Specifically, given a new question $\widetilde{Q}_o$, we first obtain its simplified sentence $\widetilde{Q}_r$, and their corresponding question skeletons $\widetilde{S}_o$ and $\widetilde{S}_r$, following the same method used in previous two subsections (see Sects. 3.1.1 and 3.1.2). Then we calculate the cosine similarity scores s_o between the semantic vector of question skeleton $\widetilde{S}_o$ and all question skeletons S in R. Similarly, we also compute the cosine similarity scores s_r for simplified question skeleton $\widetilde{S}_r$ using the formula: $s = \cos\left(\mathbf{f}(S) \cdot \mathbf{f}(\widetilde{S})\right)$, where $\mathbf{f}(.)$ represents an off-the-shelf semantic encoder[1]. Here, $\widetilde{S}$ will be instantiated as $\widetilde{S}_o$ and $\widetilde{S}_r$, and s will be instantiated as s_o and s_r, correspondingly.

From these scores, we select the top-k retrieval samples with the highest rankings. Let k_1 and k_2 denote the number of samples retrieved from the original question skeleton $\widetilde{S}_o$ and the simplified question skeleton $\widetilde{S}_r$ respectively, such that $k = k_1 + k_2$. We then concatenate the k samples to form a demonstration example as input to the LLM. Our retrieval repository offers LLMs with sample-aware SQL examples, which display a more practical answer space.

3.2   Dynamic Revision Chain

We employ LLMs to generate an initial SQL query, and then we iteratively revise the generated SQL queries based on fine-grained feedback, forming a dynamic revision chain. The dynamic revision chain consists of the SQL queries generated by the LLM iteration as nodes and the prompts provided to the LLM as edges. With minimal human intervention, LLMs interact with databases to generate accurate and executable SQL queries in two stages of the dynamic revision chain: (1) assembling prompt based on the fine-grained feedback (see Sect. 3.2.1), and (2) generating SQL via iterative prompting (see Sect. 3.2.2).

3.2.1   Fine-Grained Feedback

We collect three fine-grained pieces of information based on the SQL generated in the previous iteration. The intuition is that various information hampers LLMs' focus, so they struggle to extract necessary data from extensive and complex databases. Thus, we should progressively narrow down the scope and prioritize the most likely information. The fine-grained feedback in our approach consists of three aspects of information:

(1) **Execution Error Feedback**: We feed the SQL query generated by LLM into the database engine (i.e. SQLite) for execution. We then obtain the error messages reported during the execution and add them to the prompt. It checks whether the predicted SQL can be executed correctly, and reports the specifics of the error (e.g. *"no such table: [TABLE]"*, *"no such function: YEAR"*, *"misuse of aggregate: COUNT()"*). By incorporating the execution error messages into the prompt, LLM can learn from its errors. This helps to generate queries that follow the SQL syntax rules.

---

[1] We utilize SBERT [41] in our experiment.

(2) **Natural Language Explanation**: We prompt the LLM with instructions, converting the SQL predicted in the previous iteration back into its corresponding natural language expression. Specifically, we construct an instruction: *"What does this SQL query mean? What are the differences between the predicted meaning and the question meanings above?"*. The LLM identifies semantic gaps and fills them by explaining the meaning of its own generated SQL and comparing it to the meaning of the original question.

(3) **Related Database Contents:** We provide the LLM with content details about the database tables and columns involved in the SQL queries predicted in the previous iteration, including the possible values involved in the question. It aims to allow LLMs to simulate execution and thus generate more contextually relevant and accurate SQL queries.

Overall, the fine-grained feedback approach aims to enable LLMs to learn from their mistakes, understand the meaning of the SQL queries generated and use contextual information in the database to generate more accurate and relevant SQL queries. By addressing challenges and focusing on important aspects, the methodology aims to help the LLm better extract the necessary data from complex databases and improve the performance of its query generation.

3.2.2   Iterative SQL Generation

Based on prompts with fine-grained feedback, the LLM iteratively generates SQL queries. The intuition for iterative generation is that one iteration of fine-grained feedback might not check for all mistakes, whereas multiple iterations of feedback generation are more likely to get progressively closer to the gold answer.

Specifically, we concatenate three fine-grained feedback components with the previously generated SQL in each iteration, feeding them into the LLM. We then obtain a new SQL and collect new fine-grained feedback based on it, proceeding so to iterative generation. Let's denote the previous SQL query generated by the LLM as SQL_{prev} and the current SQL query as SQL_{curr}. The fine-grained feedback components are represented as F_{error} for execution error feedback, F_{NL} for natural language explanation, and F_{DB} for related database contents. At each iteration i, the LLM generates a new SQL query $SQL_{curr}^{(i)}$ by incorporating the fine-grained feedback components:

$$SQL_{curr}^{(i)} = \text{LLM}(SQL_{prev}, F_{error}^{(i)}, F_{NL}^{(i)}, F_{DB}^{(i)}).$$

After executing $SQL_{curr}^{(i)}$ using the database engine, we obtain the result $R_{prev}^{(i)}$ from the previous iteration and $R_{curr}^{(i)}$ from the current iteration. To avoid infinite loops, we set a maximum number of iterations N_{max}. The termination condition is defined as: $R_{prev}^{(i)} = R_{curr}^{(i)}$ or $i = N_{max}$. This control mechanism ensures that the generated SQL queries converge to an optimal and executable solution within a reasonable timeframe.

In this iterative feedback loop, we enable a dynamic interaction between the LLM and the database engine, maximizing the generation of executable SQL without extensive human intervention.

4   Experiments

4.1   Experimental Setup

4.1.1   Setting

We evaluate our method on text-davinci-003, which offers a balance between capability and availability. We apply FAISS [42] for storing the question skeletons and efficient retrieval followed by Guo et al. [35]. For the initial simplification of questions, we set temperature $\tau = 1.0$. When generating SQL samples, we set temperature $\tau = 0.5$. For the number of retrieval samples, we assign $k_1 = 4$ and $k_2 = 4$. The maximum number of iterations N is 8.

4.1.2   Datasets

(1) **Spider** [43] is a large-scale benchmark of cross-domain Text-to-SQL across 138 different domain databases. (2) **Spider-Syn** [44] is a challenging variant based on Spider that eliminates explicit alignment between questions and database schema by synonym substitutions. (3) **Spider-DK** [45] is also a variant dataset based on Spider with artificially added domain knowledge.

4.1.3   Evaluation

We consider two key metrics: execution accuracy (EX) and test-suite accuracy (TS) [46]. EX measures the accuracy of the execution results by comparing them with the standard SQL queries, while TS measures whether the SQL passes all EX evaluations for multiple tests, generated by database augmentation. Note that EX is the most direct indication of the model performance in Text-to-SQL, although it contains false positives. Exact match evaluation is not performed, as multiple correct SQLs exist for one query. We use the official TS evaluation procedure, while for EX, we slightly modify the evaluation procedure due to the need to decouple the fine-tuning-based models for independent evaluation.

4.1.4   Baselines

We compare to two groups of methods:
Fine-Tuning T5-3B Baselines: PICARD [3] is a technique that constrains auto-regressive decoders in language models through incremental parsing; **RASAT** [26], which incorporates relation-aware self-attention into transformer models while also utilizing constrained auto-regressive decoders; and **RESD-SQL** [5], which introduces a ranking-enhanced encoding and skeleton-aware decoding framework to effectively separate schema linking and skeleton parsing.

Prompting LLMs Baselines: As for the large language models, we use two variants of the Codex family [9,12] (**Davinci** and **Cushman**), **PaLM-2** [38,47], the **GPT-4** model [10,14] and the **ChatGPT** model [13]. In addition to a simple baseline assessment model, we choose several recent LLM-based works. **DIN** [14] decompose the Text-to-SQL tasks into sub-tasks: schema linking, query classification and decomposition, SQL generation, and self-correction; then performing few-shot prompting with GPT-4 [10]. **SELF-DEBUGGING** [15] adds error messages to the prompt and conducts multiple rounds of few-shot prompting for self-correction. **Few-shot SQL-PaLM** [38] adopts an execution-based self-consistency prompting approach.

4.2   Main Results

4.2.1   Performance on Spider Dataset

Table 1 shows how ours performed on Spider compared to baseline methods. Across all three datasets, our methods achieve the highest of execution accuracy (EX) and test suite accuracy (TS).

Table 1. Performance comparison on Spider with various methods. "-" indicates that the results are not available. Schema indicates that the prompt contains the SQL for creating the database tables (i.e. tables, columns, value and its type)[4].

Model	Method	EX	TS
T5-3B	T5-3B + PICARD [3]	79.3	69.4
	RASAT + PICARD [26]	80.5	70.3
	RESDSQL-3B + NatSQL [5]	84.1	73.5
Codex-cushman	Few-shot [12]	61.5	50.4
	Few-shot + Schema [12]	63.7	53.0
Codex-davinci	Few-shot [12]	60.8	51.2
	Few-shot + Schema [12]	67.0	55.1
	Few-shot [14]	71.0	61.5
	DIN-SQL [14]	75.6	69.9
	SELF-DEBUGGING [15]	84.1	-
PaLM2	Few-shot SQL-PaLM [38]	82.7	77.3
	Fine-tuned SQL-PaLM [38]	82.8	78.2
GPT-4	Zero-shot [14]	72.9	64.9
	Few-shot [14]	76.8	67.4
	DIN-SQL [14]	82.8	74.2
ChatGPT	Zero-shot [13]	70.1	60.1
Text-davinci	Zero-shot	73.1	71.6
	Ours	**85.0 (0.9↑)**	**83.2 (5.9↑)**

Ours exhibits strong performance on test-suite accuracy, which exceeds the next-best method results in fine-tuning and prompting by 9.7% and 5.9% respectively. In terms of EX, ours outperforms the next best method in both fine-tuning and prompting by 0.9%.

Comparison with Zero-Shot Prompting Models: Across all three metrics, ours surpasses Codex, ChatGPT and even GPT-4 models utilizing zero-shot prompting, despite they employ the prescribed format as outlined in the official guidelines[5]. This indicates that although LLMs are trained using a specific format, their acquired competencies become internalized and subsequently expanded for application within more flexible formats.

Comparison with Few-Shot Prompting Models: Ours also outperforms all models in a few-shot setting. The closest EX performance compared to ours is SELF-DEBUGGING, with an iterative prompting strategy as well, but we still outperform it by 0.9%. Notice that the two methods with similar few-shot prompting in the Codex-davinci model, the latter performs 10% better than the former in both EX and TS. It indicates that the selection of demonstration examples (easy, non-nested complex, and nested complex classes) [14] plays a significant role. While ours uses an adaptive sample-aware method brings 8.2% more effective than this static demonstration, which suggests that incorporating more effective prompts is crucial for LLMs to understand new specific tasks.

4.2.2   Performance on Spider-SYN and DK Datasets
Table 2 shows that ours is significantly more robust than baseline methods for Spider variants. As compared to Spider-SYN, ours improved 4.5% on EX and 12.6% on TS. Surprisingly, ours improved by 13.6% over the previous SOTA on Spider-DK.

4.3   Various Difficulty Levels Analysis

As shown in Table 3, we evaluate our effectiveness at various difficulty levels, which are determined by the number of SQL keywords used, the presence of nested sub-queries, and the utilization of column selections or aggregations. The results show that ours outperforms the other models at all levels except for the easy level, where it is worse than SQL-PaLM. The improvement in performance with increasing difficulty levels indicates that our model's strengths become more pronounced as the queries become more challenging. This suggests that our model excels in handling complex SQL queries.

4.4   Ablation Study

Figure 2 demonstrates with and without each of the two modules at four complexity levels. It shows that the exclusion of any of the modules leads to a

---

[5] https://platform.openai.com/examples/default-sqltranslate.

Table 2. Evaluation of our method on Spider-SYN and Spider-DK datasets[10]. "-" indicates that the results are not available[11].

SPIDER-SYN			
Method		**EX**	**TS**
Fine-tuning	T5-3B + PICARD [3]	69.8	61.8
	RASAT + PICARD [26]	70.7	62.4
	RESDSQL-3B + NatSQL [5]	76.9	66.8
Prompting	ChatGPT [13]	58.6	48.5
	Text-davinci	60.7	60.3
	Few-shot SQL-Palm [38]	74.6	67.4
	Fine-tuned SQL-Palm [38]	70.9	66.4
	Ours	**81.4(4.5↑)**	**80.0(12.6↑)**
SPIDER-DK			
Method		**EX**	**TS**
Fine-tuning	T5-3B + PICARD [3]	62.5	-
	RASAT + PICARD [26]	63.9	-
	RESDSQL-3B + NatSQL [5]	66.0	-
Prompting	ChatGPT [12]	62.6	-
	Text-davinci	66.2	-
	Few-shot SQL-Palm [38]	66.5	-
	Fine-tuned SQL-Palm [38]	67.5	-
	Ours	**81.1 (13.6↑)**	-

Table 3. Test-suite accuracy at various complexity levels on Spider. The first four rows are from [38], which is described as execution accuracy data in the original paper [14], but actually, it is test-suite accuracy data.

Prompting	Easy	Medium	Hard	Extra	All
Few-shot (CodeX-davinci) [14]	84.7	67.3	47.1	26.5	61.5
Few-shot (GPT-4) [14]	86.7	73.1	59.2	31.9	67.4
DIN-SQL (CodeX-davinci) [14]	89.1	75.6	58.0	38.6	69.9
DIN-SQL (GPT-4) [14]	91.1	79.8	64.9	43.4	74.2
Few-shot SQL-PaLM (PaLM2) [38]	**93.5**	84.8	62.6	48.2	77.3
Fine-tuned SQL-PaLM (PaLM2) [38]	**93.5**	85.2	68.4	47.0	78.2
Ours (Text-davinci)	91.9	**88.6**	**75.3**	**63.9**	**83.2**

decrease in performance at all levels of difficulty, in terms of hard and extra levels. Decreases in model performance are similar for the w/o revise and w/o simplify settings. Note that both modules of our method are most effective in improving the Spider-DK's easy level by 13.6% each, which requires additional domain knowledge. This suggests that the simplification strategy and dynamic revision chain strategy contribute to a variety of generalisation issues.
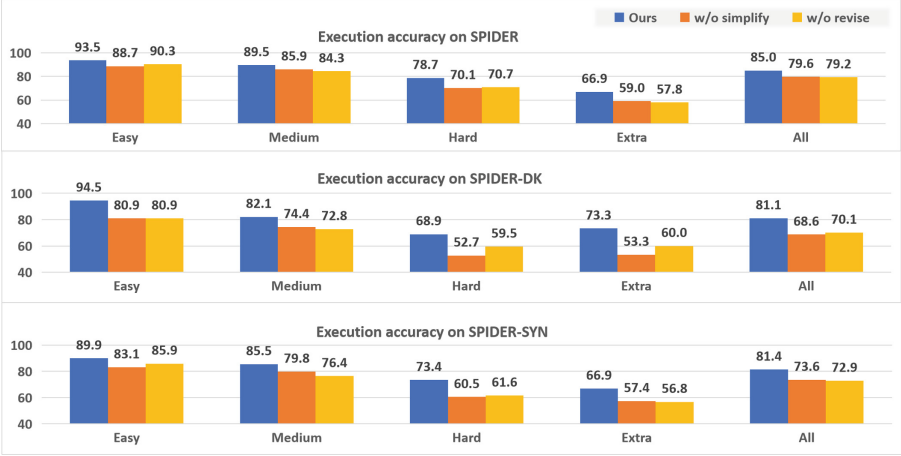

Fig. 2. Ablation study of our model components at various complexity levels across three datasets. w/o simplify refers to using a direct retrieval of the question skeletons rather than a strategy of simplifying questions. w/o revise refers to removing the dynamic revision chain module.

We found that removing the simplification module resulted in a significant drop in model performance, particularly in the DK dataset where the overall drop was 12.5%. The impact at different difficulty levels is in descending order of extra, hard, easy, and medium. This is possibly due to the fact that the model can incorporate more external knowledge as a supplementary description when simplifying, especially in the case of more SQL components. Note that w/o simplify is rather more effective for solving easy-level problems than medium-level ones, probably because the execution accuracy of easy-level problems is already high and short sentences are more likely to cause ambiguity.

Without the revision module, model performance suffers more as the difficulty level increases. On Spider-DK the model performance decreases by 11.0%, especially on easy-level and extra-level by 13.6% and 13.3% respectively. As higher difficulty levels require more knowledge, this suggests that the fine-grained feedback in the revision module effectively complements the domain knowledge required for SQL generation.

4.5   Iterative Round Analysis

From Fig. 3, we observe that the major improvement comes from the first two iteration turns. We note that in addition to the 4.6% improvement in the first iteration of Spider, the other two datasets studied for generalisability, Spider-DK and Spider-SYN, also showed a slight improvement in accuracy in the second iteration over the first. This indicates that iterative feedback of fine-grained information from a dynamic revision chain helps to deal with more complex generalisation problems, comparable to multiple reasoning needs to progressively derive the target answer.
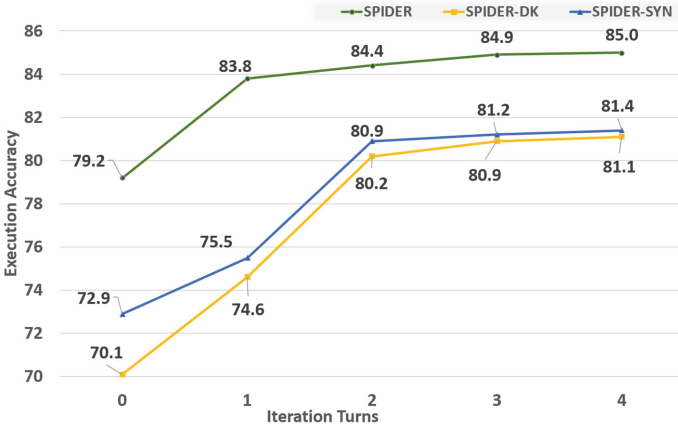
Fig. 3. Analysis of dynamic SQL revision chain with different numbers of iteration rounds across three datasets: Spider, Spider-SYN, and Spider-DK.

4.6   Case Study

To demonstrate our model, we show a comparison of predicted SQL queries in Fig. 4 using ChatGPT [12], DIN-SQL [14], SQL-PaLM [38] and Ours.

In the first example, since the question obviously mentions "*French*", the general models will be confused about the exact value of the column "*citizenship*" even if they pick it out. Noting that a SQL query must match the exact word mentioned to find the correct answer. Our approach provides the exact value of the database content involved in the first fine-grained iteration, which leads to a golden answer.

The second example requires only the selection of one item, whereas DIN-SQL and SQL-PaLM both select two. ChatGPT incorrectly uses the aggregate function *COUNT()*, which in this case is required in conjunction with *GROUP BY*. Our approach self-corrects the error in the second fine-grained iteration by interpreting the SQL interpretation in natural language.

Question:	What are the names of the singers who are not French citizens?	
ChatGPT :	SELECT Name FROM singer WHERE Citizenship <> 'French'	✘
DIN-SQL:	SELECT Name FROM singer WHERE Citizenship != 'French'	✘
SQL-PaLM:	SELECT name FROM singer WHERE citizenship ! = "French"	✘
Ours:	SELECT singer.Name FROM singer WHERE singer.Citizenship != "France"	√
Question:	Please show the record formats of orchestras in ascending order of count.	
ChatGPT :	SELECT Orchestra, Major_Record_Format FROM orchestra ORDER BY COUNT(*) ASC (Error feedback: misuse of aggregate: COUNT())	✘
DIN-SQL:	SELECT Major_Record_Format, COUNT(*) as count FROM orchestra GROUP BY Major_Record_Format ORDER BY count ASC	✘
SQL-PaLM:	SELECT major_record_format , count(*) FROM orchestra GROUP BY major_record_format ORDER BY count(*) Asc	✘
Ours:	SELECT Orchestra.Major_Record_Format FROM Orchestra GROUP BY Orchestra.Major_Record_Format ORDER BY COUNT(*) ASC	√

Fig. 4. Two illustrative cases from Spider [43]. Blue-coloured text is the correct generation, while the red-coloured text indicates the wrong generation. On the right hand side, √ means correct SQL while × means wrong. (Color figure online)

5   Conclusion

We propose retrieval-augmented prompts for an LLM-based Text-to-SQL model. By utilizing sample-aware prompting and a dynamic revision chain, we address the challenge of retrieving helpful examples and adapting the generated SQL based on fine-grained feedback. Experimental results on three Text-to-SQL benchmarks demonstrate the effectiveness of our method.

References

1. Wang, B., Shin, R., Liu, X., Polozov, O., Richardson, M.: RAT-SQL: relation-aware schema encoding and linking for text-to-SQL parsers. ACL (2020)
2. Cai, R., Xu, B., Zhang, Z., Yang, X., Li, Z., Liang, Z.: An encoder-decoder framework translating natural language to database queries. In: IJCAI, July 2018 (2018)
3. Scholak, T., Schucher, N., Bahdanau, D.: PICARD: parsing incrementally for constrained auto-regressive decoding from language models. In: EMNLP (2021)
4. Cai, R., Yuan, J., Xu, B., Hao, Z.: SADGA: structure-aware dual graph aggregation network for text-to-SQL. In: NIPS, December 2021 (2021)
5. Li, H., Zhang, J., Li, C., Chen, H.: Decoupling the skeleton parsing and schema linking for text-to-SQL. arXiv arXiv:2302.05965 (2023)
6. Li, J., Hui, B., et al.: Graphix-T5: mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing. arXiv arXiv:2301.07507 (2023)
7. Zhao, W.X., Zhou, K., Li, J., et al.: A survey of large language models. arXiv preprint arXiv:2303.18223 (2023)
8. Brown, T., Mann, B., Ryder, N., et al.: Language models are few-shot learners. In: NIPS, vol. 33, pp. 1877–1901 (2020)
9. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H.P., et al.: Evaluating large language models trained on code. arXiv arXiv:2107.03374 (2021)

10. OpenAI: GPT-4 technical report (2023)
11. Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G.: Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing. ACM Comput. Surv. **55**(9), 1–35 (2023)
12. Rajkumar, N., Li, R., Bahdanau, D.: Evaluating the text-to-SQL capabilities of large language models. arXiv arXiv:2204.00498 (2022)
13. Liu, A., Hu, X., Wen, L., Yu, P.S.: A comprehensive evaluation of ChatGPT's zero-shot text-to-SQL capability. arXiv arXiv:2303.13547 (2023)
14. Pourreza, M., Rafiei, D.: DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. arXiv preprint arXiv:2304.11015 (2023)
15. Chen, X., Lin, M., Schärli, N., Zhou, D.: Teaching large language models to self-debug. arXiv preprint arXiv:2304.05128 (2023)
16. Song, Y., et al.: Retrieval bias aware ensemble model for conditional sentence generation. In: ICASSP, pp. 6602–6606. IEEE (2022)
17. Min, S., et al.: Rethinking the role of demonstrations: what makes in-context learning work? arXiv preprint arXiv:2202.12837 (2022)
18. Guo, J., et al.: Towards complex text-to-SQL in cross-domain database with intermediate representation. arXiv preprint arXiv:1905.08205 (2019)
19. Bogin, B., Berant, J., Gardner, M.: Representing schema structure with graph neural networks for text-to-SQL parsing. In: ACL, September 2019 (2019)
20. Chen, Z., et al.: ShadowGNN: graph projection neural network for text-to-SQL parser. In: NAACL, June 2021 (2021)
21. Cao, R., Chen, L., et al.: LGESQL: line graph enhanced text-to-SQL model with mixed local and non-local relations. In: ACL, July 2021 (2021)
22. Hui, B., Geng, R., Wang, L., et al.: S^{2}SQL: injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers. In: ACL, pp. 1254–1262 (2022)
23. Hui, B., Geng, R., Ren, Q., et al.: Dynamic hybrid relation exploration network for cross-domain context-dependent semantic parsing. In: AAAI, May 2021 (2021)
24. Yu, T., Zhang, R., Polozov, A., Meek, C., Awadallah, A.H.: SCore: pre-training for context representation in conversational semantic parsing. In: ICLP (2021)
25. Cai, Z., Li, X., Hui, B., Yang, M., Li, B., et al.: STAR: SQL guided pre-training for context-dependent text-to-SQL parsing. In: EMNLP, October 2022 (2022)
26. Qi, J., Tang, J., He, Z., et al.: RASAT: integrating relational structures into pre-trained Seq2Seq model for text-to-SQL. In: EMNLP, pp. 3215–3229 (2022)
27. Lin, X.V., Socher, R., Xiong, C.: Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In: EMNLP, pp. 4870–4888 (2020)
28. He, P., Mao, Y., Chakrabarti, K., Chen, W.: X-SQL: reinforce schema representation with context. arXiv arXiv:1908.08113 (2019)
29. Lyu, Q., Chakrabarti, K., Hathi, S., Kundu, S., Zhang, J., Chen, Z.: Hybrid ranking network for text-to-SQL. arXiv preprint arXiv:2008.04759 (2020)
30. Zhong, V., Lewis, M., Wang, S.I., Zettlemoyer, L.: Grounded adaptation for zero-shot executable semantic parsing. In: EMNLP, pp. 6869–6882 (2020)
31. Choi, D., Shin, M.C., Kim, E., Shin, D.R.: RYANSQL: recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. In: CL (2021)
32. Guo, J., et al.: Towards complex text-to-SQL in cross-domain database with intermediate representation. In: ACL, pp. 4524–4535 (2019)
33. Brown, T., Mann, B., et al.: Language models are few-shot learners. In: NIPS, vol. 33, pp. 1877–1901 (2020)
34. Wei, J., et al.: Chain-of-thought prompting elicits reasoning in large language models. In: Advances in Neural Information Processing Systems, vol. 35, pp. 24824–24837 (2022)

35. Guo, C., et al.: A case-based reasoning framework for adaptive prompting in cross-domain text-to-SQL. arXiv preprint arXiv:2304.13301 (2023)
36. Zhou, D., et al.: Least-to-most prompting enables complex reasoning in large language models. arXiv preprint arXiv:2205.10625 (2022)
37. Tian, Y., Li, T.J.J., Kummerfeld, J.K., Zhang, T.: Interactive text-to-SQL generation via editable step-by-step explanations. arXiv preprint arXiv:2305.07372 (2023)
38. Sun, R., et al.: SQL-PaLM: improved large language model adaptation for text-to-SQL. arXiv arXiv:2306.00739 (2023)
39. Tian, Z., Bi, W., Li, X., Zhang, N.L.: Learning to abstract for memory-augmented conversational response generation. In: ACL, pp. 3816–3825 (2019)
40. Wen, Z., et al.: GRACE: gradient-guided controllable retrieval for augmenting attribute-based text generation. In: Findings of ACL 2023, pp. 8377–8398 (2023)
41. Reimers, N., Gurevych, I.: Sentence-BERT: sentence embeddings using Siamese BERT-networks. In: EMNLP-IJCNLP, pp. 3982–3992. ACL (2019). https://aclanthology.org/D19-1410
42. Johnson, J., Douze, M., Jegou, H.: Billion-scale similarity search with GPUs. IEEE Trans. Big Data **7**, 535–547 (2019)
43. Yu, T., Zhang, R., et al.: Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In: EMNLP, June 2019 (2019)
44. Gan, Y., Chen, X., Huang, Q., Purver, M., et al.: Towards robustness of text-to-SQL models against synonym substitution. In: ACL, July 2021 (2021)
45. Gan, Y., Chen, X., Purver, M.: Exploring underexplored limitations of cross-domain text-to-SQL generalization. In: EMNLP, December 2021 (2021)
46. Zhong, R., Yu, T., Klein, D.: Semantic evaluation for text-to-SQL with distilled test suites. In: EMNLP, pp. 396–411 (2020)
47. Anil, R., et al.: PaLM 2 technical report. arXiv preprint arXiv:2305.10403 (2023)