Prompting GPT-3.5 for Text-to-SQL with De-semanticization and Skeleton Retrieval

Chunxi Guo, Zhiliang Tian(✉), Jintao Tang(✉), Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang(✉)

College of Computer, National University of Defense Technology, Changsha, China
{chunxi,tianzhiliang,tangjintao,wangpancheng13,zhwen,yangkang, tingwang}@nudt.edu.cn

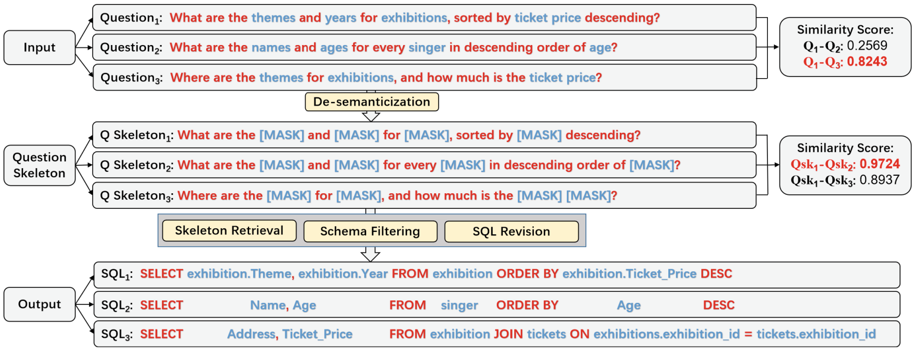
Abstract. Text-to-SQL is a task that converts a natural language question into a structured query language (SQL) to retrieve information from a database. Large language models (LLMs) work well in natural language generation tasks, but they are not specifically pre-trained to understand the syntax and semantics of SQL commands. In this paper, we propose an LLM-based framework for Text-to-SQL which retrieves helpful demonstration examples to prompt LLMs. However, questions with different database schemes can vary widely, even if the intentions behind them are similar and the corresponding SQL queries exhibit similarities. Consequently, it becomes crucial to identify the appropriate SQL demonstrations that align with our requirements. We design a de-semanticization mechanism that extracts question skeletons, allowing us to retrieve similar examples based on their structural similarity. We also model the relationships between question tokens and database schema items (i.e., tables and columns) to filter out scheme-related information. Our framework adapts the range of the database schema in prompts to balance length and valuable information. A fallback mechanism allows for a more detailed schema to be provided if the generated SQL query fails. Ours outperforms state-of-the-art models and demonstrates strong generalization ability on three cross-domain Text-to-SQL benchmarks.

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

1 Introduction

Text-to-SQL tasks aim to transform natural language questions (NLQ) into structured query language (SQL), enabling users without expertise in database querying to retrieve information from a database [1,2]. Considering that databases are used in various scenarios involving different domains (e.g., education, financial systems), researchers have adapted encoder-decoder architecture [3,4], which eliminates the need for domain-specific knowledge through end-to-end training. To train the model, these approaches require diverse and extensive training data, which can be prohibitively expensive [5].

Large pre-trained language models (LLMs) (e.g., GPT-3 [6] and Codex [7]) encompass more extensive data and parameters than traditional pre-trained language models (e.g., BERT [8], RoBERTa [9], BART [10] and T5 [11]) and exhibit superior performance on a variety of tasks, including Text-to-SQL. Rajkumar et al. [12] and Liu et al. [13] evaluate LLMs' performance in Text-to-SQL in zero- and few-shot settings. Cheng et al. [14] present a neural-symbolic framework that maps the input to a program, which incorporates symbolic components into LLMs. However, many studies found that LLMs perform worse than traditional non-LLM-based approaches in Text-to-SQL [3,15,16]. As the existing LLMs are not designed for understanding the syntax and semantics of SQL commands, it is challenging for them to accurately generate complex SQL commands (e.g. *SELECT*, *WHERE*, *AVG*, *DESC*). To accurately map out these SQL commands, it is essential to distinguish the question intention. Intention in Text-to-SQL tasks refers to the collection of query-related specifications and directives that encompass the desired scope, criteria, and actions to be performed on a database. This concept encompasses various desired result set attributes, including data volume, sorting sequence, and filtering prerequisites. For example, the skeleton corresponds to the question *"What are the names of the singers who are not French?"* is *"What are the [MASK] of the [MASK] who are not [MASK]?"*, whose intention is to query a term with a conditional constraint.



Input
Question1: What are the themes and years for exhibitions, sorted by ticket price descending?
Question2: What are the names and ages for every singer in descending order of age?
Question3: Where are the themes for exhibitions, and how much is the ticket price?
Similarity Score:
Q1-Q2: 0.2569
Q1-Q3: 0.8243
De-semanticization
Question Skeleton
Q Skeleton1: What are the [MASK] and [MASK] for [MASK], sorted by [MASK] descending?
Q Skeleton2: What are the [MASK] and [MASK] for every [MASK] in descending order of [MASK]?
Q Skeleton3: Where are the [MASK] for [MASK], and how much is the [MASK] [MASK]?
Similarity Score:
Qsk1-Qsk2: 0.9724
Qsk1-Qsk3: 0.8937
Skeleton Retrieval
Schema Filtering
SQL Revision
Output
SQL1: SELECT exhibition.Theme, exhibition.Year FROM exhibition ORDER BY exhibition.Ticket_Price DESC
SQL2: SELECT Name, Age FROM singer ORDER BY Age DESC
SQL3: SELECT Address, Ticket_Price FROM exhibition JOIN tickets ON exhibitions.exhibition_id = tickets.exhibition_id

Fig. 1. Comparison of three examples (i.e. question, question skeleton, SQL). The first example is similar to the second in terms of question intention (sorting), and to the third in terms of vocabulary of the questions. Note that the intention of the third question is to get two attribute items and there may be a table join. We aim to obtain SQL queries with the same commands (i.e. *ORDER BY*, *DESC*) in the prompt. Compared with the full question similarity score, the question skeleton increases the absolute value as well as the relative ranking.

LLMs are proven to fast adapt to new paradigms with few-shot examples [17–19]. We argue that LLMs probably quickly learn to follow some demonstration examples of SQL generation, even if the SQL generation involves multiple SQL commands and nested clauses.

In this paper, we propose an LLM-based Text-to-SQL framework that retrieves a few demonstration examples to prompt the LLM according to the skeleton of the input question. Notice that questions with different database schemes may be distinct since questions contain much scheme-related information (i.e. the blue texts in Fig. 1's upper side), even if they have similar intention and SQL queries. It can be difficult for the model to retrieve helpful examples. To solve this issue, we design a de-semanticization mechanism to extract skeletons of questions. We retrieve similar SQL demonstrations, which share similar question skeletons with the input question. Besides, during de-semanticization, we model the relationships between question tokens and scheme items to filter out the scheme items related to the question tokens. In this way, we achieve schema linking concerning the question-scheme relevance. Finally, we adaptively control the range of the database schema in prompts to balance length and valuable information. Through a fallback mechanism, the model receives a more detailed schema if the generated SQL query needs revision. Our framework excels compared to commercial Text-to-SQL engines like AI2sql[1] by emphasizing transparent, flexible algorithms for better performance and deeper insights into the conversion process, whereas comparisons to large models like GPT-4 might not be entirely fair due to data volume discrepancies.

Our contributions are as follows: (1) We propose an LLM-based framework for Text-to-SQL tasks that retrieves similar examples to augment prompts for LLMs. (2) We design a de-semanticization mechanism that effectively removes scheme-related information for retrieving texts with similar skeletons. (3) Our method surpasses the SOTA models and exhibits strong generalization.

2 Related Work

The evolution of SQL generation techniques showcases a progression from encoder-decoder architectures [2] to LLM-based solutions.

Encoder-Based SQL Generation. Guo et al. [20] introduced IRNET, utilizing attention-based Bi-LSTM to encode and an intermediate representation-based decoder for SQL prediction. Afterwards, graph-based encoders were integrated to enhance input representations [21,22]. Works such as RATSQL [1], LGESQL [23], R2SQL [24], SDSQL [3], S2SQL [25], and STAR [26] focused on refining structural reasoning by explicitly modeling relationships between schemas and questions. Notably, GRAPHIX-T5 [4] overcame prior limitations by incorporating graph representation learning into the encoder. Simultaneously, RASAT [16] augmented T5 with structural insights by introducing edge embeddings into multi-head self-attention.

Decoder-Based SQL Generation. We categorize the methods into four distinct groups: Sequence-based methods like BRIDGE [27] and PICARD [15] directly translate natural language queries into SQL queries token by token. Template-based methods, represented by X-SQL [28] and HydraNet [29], utilize predefined templates to guide SQL generation, ensuring structural coherence. Stage-based methods, exemplified by GAZP [30] and RYANSQL [31],

[1] https://app.ai2sql.io.

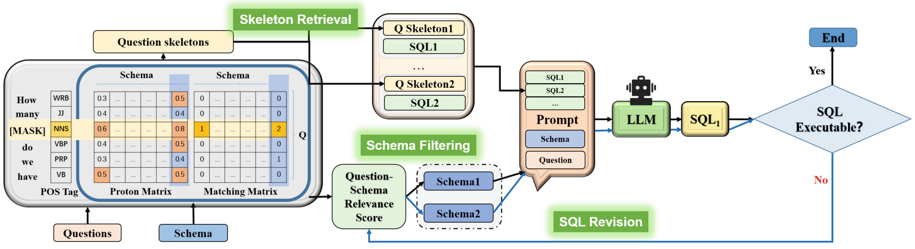


Fig. 2. The overview of our framework. The grey box on the left shows the details of the de-semantization process. The rest is the process of prompt construction. We revise the SQL following the blue line. (Color figure online)

involve establishing a coarse-grained SQL framework, subsequently employing slot-filling methodologies to complete missing details within the framework. Lastly, hierarchical-based methods, such as IRNet [20] and RAT-SQL [1], adopt a hierarchical approach to tackle NLQ-to-SQL translation.

LLM-Based SQL Generation. Recently, LLM-based models are now prominent options for this task. Unlike full-data fine-tuned models, LLM-based models can achieve good performance with just a few unsupervised in-context exemplar annotations [5]. Yu et al. [32] introduced a method to classify and cluster SQL queries based on question characteristics. Inspired by some retrieval-related research [33–35], we retrieve SQL examples with the same intention as a demonstration, thereby enhancing the comprehension of the diverse operators and their respective applications. Our approach improves the LLM's performance to generate valid and accurate SQL queries.

3 Methodology

Our framework consists of two modules as shown in Fig. 2: (1) **Question De-semanticization** (Sect. 3.1) removes tokens that are semantically related to the domain and preserves the question skeletons, which represent the question's intentions. (2) **LLM-Based Adjustable Prompting** (Sect. 3.2) involves using the SQL demonstrations with the same intention and corresponding database schema to create prompts that guide the LLM in generating SQL queries.

Given a natural language question Q and the database schema $S = \langle T, C \rangle$, the goal of Text-to-SQL tasks is to generate the corresponding SQL P. Here the question $Q = \left(q_1, q_2, \ldots, q_{|Q|}\right)$ is a sequence of words. The database schema consists of tables $T = \left(t_1, t_2, \cdots, t_{|T|}\right)$, and columns $C = \left(c_1, c_2, \cdots, c_{|C|}\right)$.

3.1 Question De-semanticization

We remove question tokens that are semantically related to the database schema (i.e., table, column) and obtain the question skeletons, which represent the question's intentions. In this way, from a matching perspective, by eliminating the schema-related information from the question, we can better match examples with similar question intent when retrieving.

There are two steps involved in this process: the first step is schema linking, which finds the question tokens related to the schema. Meanwhile, we obtain the relevance of the question tokens to the schema. As a second step, we mask the tokens and obtain the skeletons of the questions. In the first step, we cannot determine how relevant the tokens are to the schema, so we design schema-related detection (Sect. 3.1.1), token matching (Sect. 3.1.2) and part-of-speech tagging (Sect. 3.1.3) strategies to find relevant tokens.

3.1.1 Schema-Related Detection

We use a masking technique to identify correlations between question words and their corresponding database schema. Concretely, we calculate the similarity between question tokens and schema items. There are three steps:

(1) Masking and Concatenation. We concatenate a question Q and schema (i.e., table T, column C) into one long sequence. We also mask each token of the question in the sequence to generate a series of sequences. Formally, we obtain a series of sequences as follows, where [CLS] and [SEP] denote classification and sentence separation, respectively:

$$[CLS]q_1q_2\cdots q_{|Q|}[SEP][CLS]t_1\cdots t_{|T|}[SEP][CLS]c_1\cdots,c_{|C|}[CLS]$$

$$[CLS][MASK]q_2\cdots q_{|Q|}[SEP][CLS]t_1\cdots,t_{|T|}[SEP][CLS]c_1\cdots,c_{|C|}[CLS]$$

$$\cdots$$

Later, we put all sequences into a pre-trained language model to obtain deep contextualized representations. We denote h_j^s as the representation of schema item s_j and $h_{j\setminus q_i}^s$ as the representation if the question token q_i is masked out.

(2) Representation Transformation. We utilize the standard Poincaré ball [36]-a unique model of hyperbolic spaces to project the representations. We get the hyperbolic representations, denoted as $\tilde{h}$, using the following method:

$$\tilde{\mathrm{h}} = g_0(\mathrm{~h}) = \tanh(\|\mathrm{h}\|)\frac{\mathrm{h}}{\|\mathrm{h}\|}. \tag{1}$$

The hyperbolic space provides a suitable geometry for modeling the hierarchy [36]. The hyperbolic space has a property called a negative curve which allows for a more efficient representation of the hierarchy, enabling our method to capture long-term dependencies and the overall sentence structure [37]. Furthermore, the hyperbolic space has a greater power to represent than the Euclidean space low-dimensional space, thus allowing for a more efficient representation of sentences with semantic hierarchy.

(3) Correlation Measurement. We measure the correlation between the question token q_i and the schema item s_j in the hyperbolic space. Concretely, we elicit the correlation between question tokens and schema items from a pre-trained language model based on the Poincaré distance matrix [37].

By computing d_p on each pair of tokens (q_i, s_j), we get the Proton matrix $\mathrm{D_p} \in \mathbb{R}^{|Q|\times|S|}$ as follows:

$$\mathrm{D_p}_{i,j} = 2\tanh^{-1}\left(\left\|-\tilde{h}^s_{j\backslash q_i} \oplus \tilde{h}^s_j\right\|\right), \tag{2}$$

where $\oplus$ is the Möbius addition [36], $\tilde{h}^s_j$ represents the embedding of the schema item s_j, and $\tilde{h}^s_{j\backslash q_i}$ represents the embedding if the question token q_i is masked out. Both $\tilde{h}^s_j$ and $\tilde{h}^s_{j\backslash q_i}$ are hyperbolic representation defined in Eq. (1).

We argue that the sequence concatenating all tables and columns represents the domain knowledge of the example, which consists of the vocabulary of the domain/scenario. We mask each token in the question sequence to detect whether it is relevant to the domain. This measure works by masking the token and checking if it causes a significant shift in the vector representation of the entire sequence. If the shift is beyond a pre-defined threshold, we consider the masked token to be important for the sequence. This means that the token is strongly correlated with the meaning expressed by the sequence.

3.1.2 Token Matching

To discover the question tokens closely related to the scheme, we match each question token and each schema item (i.e., table names, column names, and values) with two kinds of explicit information in the input: name-based and value-based matching.

Name-based matching identifies direct lexical matches between each question token and each schema item. If a sub-sequence of the question sequence $q_{i\ldots j}$ matches the schema names s_j, score one point for matching similarity. While value-based matching detects possible value correspondences within the query. If the question word q_i is equal to specific values v_j in the database, where v_j represents the set of values in the j^{th} column of the corresponding table or column, score one point for the corresponding matching similarity. We define matrix $M_m \in \mathbb{R}^{|Q|\times|S|}$ to represent the question-schema matching similarity:

$$M_{m_{i,j}} = \begin{cases} 2, & q_{i\ldots j} \subseteq s_j \wedge q_i = v_j \\ 1, & q_{i\ldots j} \subseteq s_j \vee q_i = v_j \\ 0, & \text{otherwise} \end{cases}. \tag{3}$$

So far, coupled with the previously calculated Proton matrix $\mathrm{D_p}$, we get the question-schema relevance score, which measures the probability that the schema items will be used to compose the SQL query. We define the relevance score matrix $R \in \mathbb{R}^{|Q|\times|S|}$ as follows:

$$R = \mathrm{D_p} + \beta \cdot M_m, \tag{4}$$

where β determines the relative influence of these two strategies.

3.1.3 Part-of-Speech Tagging

We perform part-of-speech (POS) tagging on questions to improve the recognition of the question skeleton.

Formally, we tag the question Q with POS analysis to obtain a set of POS tags $t_1, t_2, \ldots, t_n$, where t_i is the POS tag of a token q_i. Then we generate the lexical matrix $P \in \mathbb{R}^{|Q|}$ for each token q_i based on its POS tag t_i as follows:

(1) If t_i is a noun or a number, then P_i is assigned the value α.
(2) Otherwise, P_i is assigned the value 0.

POS information is crucial for constructing a question skeleton as it aids in comprehending the sentence's structure and the grammatical roles of the words.

Incorporating the three strategies mentioned above, we obtain a question relevance score $Q_{\mathrm{sco}\,i}$ for each question token q_i using the following equation:

$$Q_{\mathrm{sco}\,i} = \frac{1}{2}\left(\frac{1}{n}\sum_{j=1}^{|S|} R_{ij} + P_i\right), \quad \forall i \in 1, \ldots, |Q|. \tag{5}$$

We generate the question skeleton based on Q_{sco} and τ, where τ is a hyperparameter that controls the minimum relevance score required for a token.[2]

After calculating the R (in Sect. 3.1.2) and P, we remove domain-relevant tokens of the questions and generate the de-semanticized question skeletons, which allows for better retrieval of examples where the question intentions are more consistent and more applicable to the in-context demonstration.

3.2 LLM-Based Adjustable Prompting

To construct prompts for the LLM to generate new SQL queries, we utilize the SQL queries obtained from the question skeleton and the relevant database schema, which are filtered by the question-schema relevance score. Then we revise the SQL queries via a fallback mechanism, which adjusts the schema range. The prompt we designed consists of three parts as shown in Fig. 2.

3.2.1 *k*NN-Based Skeleton Retrieval

We retrieve k-NN examples based on the new question skeleton. We project de-semanticized question skeletons into a vector space and retrieve k-NN examples corresponding to the new question skeleton. We use cosine similarity to measure the text vectors. The new question skeleton serves as the key for the retrieval process, and the returned value consists of the k-NN examples. Questions with the same intention can aid in generating SQL queries by sharing common structures and requiring comparable SQL queries to extract information from a database. Recognizing patterns and similarities between questions enables the language model to produce suitable SQL queries for a given inquiry.

[2] If q_{sco} is below the threshold of τ, we retain the original question token; otherwise, we replace it with the pre-defined [MASK] token.

3.2.2 Schema-Relevance Filtering

We utilize the filtered database schema items to generate SQL queries. We get the schema with a scaled-down range by the relevance scores between the question and the schema. We apply a threshold θ to filter out less relevant schema items based on the question-schema relevance score obtained in Sect. 3.1. Specifically, we only consider schema items with scores higher than the θ. with a scaled-down schema range, we prompt the LLM to generate the SQL queries. Narrowing the schema range prevents irrelevant schema from interfering with the model's response to the current question.

3.2.3 Fallback Revision

We propose a fallback mechanism to revise and regenerate SQL queries, in cases where the LLM outputs a message indicating SQL generation failure or when the generated SQL query cannot be executed successfully. We first check if the generated SQL query is valid and can be executed on the database. If the query fails, we retrieve the complete database schema and use it to revise the SQL query. This ensures that the revised SQL query is valid and can be executed on the database. We then pass the revised SQL query to the LLM for further processing. To avoid generating an infinite loop of fallbacks, we set a maximum number of fallback attempts. If the maximum number of fallback attempts is reached and the SQL query still cannot be generated, we terminate the process.

4 Experiment

4.1 Experimental Setup

Datasets. In our study, we perform experiments on three widely recognized benchmark datasets: (1) **Spider** [38] covers a diverse range of 138 domain databases, offering a large-scale evaluation platform. (2) **Spider-Syn** [39] is a modified version of Spider that introduces difficulty by replacing explicit question-schema alignments with synonymous phrasing. (3) **Spider-DK** [40] is an augmentation of Spider, incorporating artificial domain knowledge to further test model adaptability and comprehension.

Evaluation. Valid SQL (VA) measures the percentage of SQL queries that are executed without any errors. **Execution accuracy (EX)** measures the accuracy of the execution results by comparing them with the standard SQL query. **Test-suite accuracy (TS)** [41] measures the effectiveness of the distilled test suite in achieving high code coverage for the database through execution, which can serve as a better proxy for semantic accuracy. Note that we do not rely on the mainstream exact match accuracy metric (EM), as SQL queries that serve the same purpose may be expressed in different ways. EM is tailored to a limited style of the dataset and serves as an intermediate solution evaluation metric for Text-to-SQL tasks.

Baselines. Full-data fine-tuned models: **PICARD** [15] employs incremental parsing to constrict auto-regressive decoders in language models; **RASAT** [16] enhances transformer models with relation-aware self-attention, combined with constrained auto-regressive decoders; and **RESDSQL** [3] introduces a novel framework featuring ranking-enhanced encoding and skeleton-aware decoding. LLM-based models: We select the **ChatGPT** [13] and **Codex** [12] baseline models for our study, as they are currently the top performers in evaluating Text-to-SQL capability using LLMs. Furthermore, our approach utilizes the latest **GPT-3.5** model, text-davinci-003. We all use the best performing with prompt engineering methodologies adapted to our respective models.

Experimental Setting. We use FAISS [42] to store and retrieve question skeletons. For hyperparameter settings, we assign $k = 8$, $\alpha = 0.9$, $\beta = 0.5$, $\tau = 0.6$, and $\theta = 0.4$. Our approach expands the database content beyond the limitations of the Text-to-SQL prompt used in the OpenAI demo website[3], which only contains table and column names. We re-formatted the prompt to achieve better results, though it differs from the format used in the official data training.

4.2 Main Results

Table 1. Comparison of the performance of our model and others on three datasets ("-" indicates the results are not available. The Codex model could not be reproduced due to an invalid Codex'api. The TS metric did not apply to the Spider-DK dataset).

Models\Datasets		SPIDER			SPIDER-SYN			SPIDER-DK		
		VA	EX	TS	VA	EX	TS	VA	EX	TS
Full-data Fine-tuned Models	PICARD	98.4	79.3	69.4	98.2	69.8	61.8	97.8	62.5	-
	RASAT	98.8	80.5	70.3	98.3	70.7	62.4	98.5	63.9	-
	RESDSQL-3B	**99.1**	84.1	73.5	98.8	76.9	66.8	98.8	66.0	-
LLM-based Models	GPT-3.5	87.0	57.2	56.7	83.1	39.3	39.2	88.6	48.8	-
	Codex [12]	91.6	67.0	55.1	-	-	-	-	-	-
	ChatGPT [13]	97.7	70.1	60.1	96.2	58.6	48.5	96.4	62.6	-
	Ours	99.0	**87.8**	**84.8**	**99.0**	**79.4**	**75.9**	**99.4**	**74.2**	-

We present a comparison between LLM-based models and full-data fine-tuned models which are SOTA as shown in Table 1. Ours outperforms all models in almost all evaluation metrics, except for the Spider dataset where the VA is only 0.1 worse than the next best model (RESDSQL-3B). LLM-based models may face difficulty generating SQL queries that conform to strict syntactical and semantic rules, resulting in lower VA scores, compared with the fine-tuned models. We further investigate the effectiveness of ours and compare it to the

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

models using LLMs directly. We observe that the improper utilization of LLMs is generally less effective in generating complex SQL commands. Additionally, the regular LLM-based models are confused with selecting the appropriate schema items required. On the contrary, ours demonstrates its effectiveness in generating accurate and semantically meaningful SQL queries.

4.3 Ablation Study

We investigate the contributions of various components of ours as shown in Table 2: (1) DESEM+P, which uses Poincaré distance for schema-related detection in de-semanticization; (2) DESEM-P+E, which uses Euclidean distance for schema-related detection in de-semanticization; (3) -DESEM, which retrieves questions without de-semanticization using cosine similarity; (4) -Skeleton Retrieval, which demonstrates random examples without retrieval; (5) -Schema Filtering, which uses the full range of schema without filtering; (6) -SQL Revision, which directly outputs the generated SQL without revision.

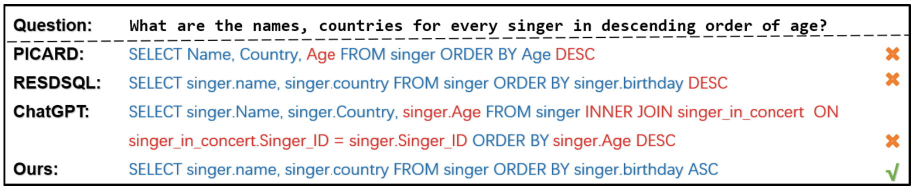
Table 2. Ablation study of different modules. "-" means not using that strategy, while "+" means using that strategy.

Methods/Datasets	SPIDER			SPIDER-SYN			SPIDER-DK		
	VA	EX	TS	VA	EX	TS	VA	EX	TS
DESEM +P (Ours)	**99.0**	**87.8**	**84.8**	**99.0**	**79.4**	**75.9**	**99.4**	**74.2**	**69.7**
DESEM -P+E	96.6	84.6	79.6	97.1	77.2	74.3	99.1	71.3	68.2
-DESEM	93.5	71.6	69.8	94.7	59.2	57.7	86.5	56.8	55.0
-Skeleton Retrieval	94.3	66.9	65.9	92.9	56.8	55.3	93.3	59.4	57.9
-Schema Filtering	96.7	79.6	78.3	97.3	73.6	71.5	98.5	68.6	64.3
-SQL Revision	98.5	81.2	78.2	95.0	69.9	67.3	97.6	71.8	67.1

Using Poincaré distance instead of Euclidean distance for schema detection (DESEM + P) improves efficacy. Removing de-semanticization (-DESEM) reduces accuracy, highlighting the importance of relevant examples for utilizing SQL components and expressing intent. Removing skeleton retrieval and comparing question sequences directly lowers performance, indicating domain knowledge's impact on the SPIDER-DK. SQL revision enhances generated SQL accuracy, as seen in higher VA scores of methods with revision. Schema filtering boosts performance by narrowing the schema range, even without revision.

4.4 Case Study

To illustrate our method, we show a comparison of predicted SQLs in Fig. 3 using PICARD, RESDSQL, ChatGPT, and our approach.



Question:	What are the names, countries for every singer in descending order of age?	
PICARD:	SELECT Name, Country, Age FROM singer ORDER BY Age DESC	✗
RESDSQL:	SELECT singer.name, singer.country FROM singer ORDER BY singer.birthday DESC	✗
ChatGPT:	SELECT singer.Name, singer.Country, singer.Age FROM singer INNER JOIN singer_in_concert ON singer_in_concert.Singer_ID = singer.Singer_ID ORDER BY singer.Age DESC	✗
Ours:	SELECT singer.name, singer.country FROM singer ORDER BY singer.birthday ASC	✓

Fig. 3. An illustrative case from Spider-DK [40]. Blue is the correct generation, and red is the wrong generation. (Color figure online)

It shows that PICARD and ChatGPT generate an extra column "Age", and the three models fall short of accurately capturing the desired sorting order. This is because the database schema only has a "birthday" column, and sorting by age is equivalent to sorting by "birthday" in ascending order. ChatGPT attempts to generate a more comprehensive SQL query by including an unnecessary JOIN operation with the singer_in_concert table. Ours considers the requirement of ordering the results, albeit in the opposite direction specified in the question. However, fine-tuned models like PICARD and RESDSQL may struggle with complex questions due to limitations in learned patterns and structures.

5 Conclusion and Future Work

We propose a Text-to-SQL generation framework that prompts LLMs with few retrieved demonstrations. A limitation of ours is over-reliance on LLMs's ability for SQL code generation. LLMs with certain capabilities can work well with our method. Future research will focus on external knowledge reasoning, as well as efficiency when dealing with large databases. Our approach can be generalized to knowledge base question answering and code generation tasks.

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 (2018)
3. Li, H., Zhang, J., Li, C., Chen, H.: Decoupling the skeleton parsing and schema linking for text-to-SQL. arXiv:2302.05965 (2023)
4. Li, J., Hui, B., et al.: Graphix-T5: mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing. arXiv:2301.07507 (2023)
5. Zhao, W.X., Zhou, K., Li, J., et al.: A survey of large language models. arXiv preprint arXiv:2303.18223 (2023)
6. Brown, T., Mann, B., Ryder, N., et al.: Language models are few-shot learners. In: NIPS, vol. 33, pp. 1877–1901 (2020)
7. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., et al.: Evaluating large language models trained on code. arXiv:2107.03374 (2021)

8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: NAACL (2018)
9. Zhuang, L., Wayne, L., Ya, S., Jun, Z.: A robustly optimized bert pre-training approach with post-training. In: CCL, pp. 1218–1227 (2021)
10. Lewis, M., Liu, Y., et al.: Bart: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: ACL (2020)
11. Raffel, C., Shazeer, N., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. JMLR **21**, 5485–5551 (2020)
12. Rajkumar, N., Li, R., Bahdanau, D.: Evaluating the text-to-SQL capabilities of large language models. 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:2303.13547 (2023)
14. Cheng, Z., Xie, T., Shi, P., et al.: Binding language models in symbolic languages. In: ICLR (2023)
15. Scholak, T., Schucher, N., Bahdanau, D.: Picard: parsing incrementally for constrained auto-regressive decoding from language models. In: EMNLP (2021)
16. Qi, J., Tang, J., He, Z., et al.: RASAT: integrating relational structures into pretrained Seq2Seq model for text-to-SQL. In: EMNLP, pp. 3215–3229 (2022)
17. Lee, Y.J., Lim, C.G., Choi, H.J.: Does GPT-3 generate empathetic dialogues? A novel in-context example selection method and automatic evaluation metric for empathetic dialogue generation. In: COLING, pp. 669–683 (2022)
18. Su, H., Kasai, J., et al.: Selective annotation makes language models better few-shot learners. arXiv:2209.01975 (2022)
19. Rubin, O., Herzig, J., Berant, J.: Learning to retrieve prompts for in-context learning. In: NAACL, pp. 2655–2671 (2022)
20. Guo, J., et al.: Towards complex text-to-SQL in cross-domain database with intermediate representation. In: ACL, pp. 4524–4535 (2019)
21. Bogin, B., Berant, J., Gardner, M.: Representing schema structure with graph neural networks for text-to-SQL parsing. In: ACL (2019)
22. Chen, Z., et al.: ShadowGNN: graph projection neural network for text-to-SQL parser. In: NAACL (2021)
23. Cao, R., Chen, L., et al.: LGESQL: line graph enhanced text-to-SQL model with mixed local and non-local relations. In: ACL (2021)
24. Hui, B., Geng, R., Ren, Q., et al.: Dynamic hybrid relation exploration network for cross-domain context-dependent semantic parsing. In: AAAI (2021)
25. Hui, B., Geng, R., Wang, L., et al.: S2SQL: injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers. In: ACL, pp. 1254–1262 (2022)
26. 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 (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: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. CL **47**(2), 309–332 (2021)

32. Yu, W., Guo, X., Chen, F., Chang, T., Wang, M., Wang, X.: Similar questions correspond to similar SQL queries: a case-based reasoning approach for text-to-SQL translation. In: Sánchez-Ruiz, A.A., Floyd, M.W. (eds.) ICCBR 2021. LNCS (LNAI), vol. 12877, pp. 294–308. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86957-1_20
33. Tian, Z., Bi, W., Li, X., Zhang, N.L.: Learning to abstract for memory-augmented conversational response generation. In: ACL, pp. 3816–3825 (2019)
34. Song, Y., et al.: Retrieval bias aware ensemble model for conditional sentence generation. In: ICASSP, pp. 6602–6606. IEEE (2022)
35. Wen, Z., et al.: Grace: gradient-guided controllable retrieval for augmenting attribute-based text generation. In: Findings of ACL 2023, pp. 8377–8398 (2023)
36. Ganea, O., Bécigneul, G., Hofmann, T.: Hyperbolic neural networks. In: NIPS (2018)
37. Chen, B., et al.: Probing bert in hyperbolic spaces. arXiv:2104.03869 (2021)
38. 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 (2019)
39. Gan, Y., Chen, X., Huang, Q., Purver, M., et al.: Towards robustness of text-to-SQL models against synonym substitution. In: ACL (2021)
40. Gan, Y., Chen, X., Purver, M.: Exploring underexplored limitations of cross-domain text-to-SQL generalization. In: EMNLP (2021)
41. Zhong, R., Yu, T., Klein, D.: Semantic evaluation for text-to-SQL with distilled test suites. In: EMNLP, pp. 396–411 (2020)
42. Johnson, J., Douze, M., Jegou, H.: Billion-scale similarity search with GPUs. IEEE Trans. Big Data **7**, 535–547 (2019)