Enhancing Chain-of-Thought Reasoning for Text-to-SQL with Effective Retrieval-Augmented Generation

Xuguang Zhu, Yong Zhang ✉, Chao Li, and Chunxiao Xing

Tsinghua University
zhuxg21@mails.tsinghua.edu.cn, zhangyong05@tsinghua.edu.cn,
li-chao@tsinghua.edu.cn, xingcx@tsinghua.edu.cn

Abstract. The problem of text-to-SQL, which aims at translating natural language questions into SQL queries, has caught increasing attention from the research communities. With the rapid advances in the era of Large Language Model (LLM), nowadays in-context learning over LLMs has become the mainstream methodology for text-to-SQL. Although there are many previous studies in this area, their performance is not so ideal since they fail to fully utilize the reasoning capacity of LLMs. In this paper, we proposed a novel framework to improve the performance of text-to-SQL by leveraging the Chain-of-Thought style (CoT) prompt strategy over LLM. We decompose the process of SQL translation into three stages and construct effective prompts to realize each of them, respectively. To further improve the performance, we adopt the Retrieval Augment Generation (RAG) strategy to include essential contextual information in the prompts to provide more useful insights for in-context learning. We conduct experiments on several benchmarking datasets and the results show that our proposed methods could outperform existing solutions by an obvious margin.

Keywords: Text-to-SQL, Large Language Model, Chain-of-Thought, Retrieval Augment Generation

1 Introduction

Given a natural language question and a database, the text-to-SQL problem aims at translating the natural language question into SQL queries based on contents of the database. It is a crucial way to facilitate non-professional users with limited knowledge about SQL to access data stored in relational databases. Consequently, text-to-SQL has become an effective way to improve the usability of database systems. And it has been widely applied to many domains, such as finance [40], business [37], entertainment [37] and online advertising [7].

There is a long stream of research efforts in the problem of text-to-SQL in the fields of database and natural language processing. Earlier studies of text-to-SQL can be divided into three categories based on their methodologies: rule-based [29], learning based [30] and template based [8] ones. Recently, advances in the era of

Large Language Models (LLMs) have brought new opportunities to and have become the mainstream methodology for the text-to-SQL problem. LLMs such as GPT-4 [24], LLaMA [32] and CodeX [1] are generative models that take sequences as input and generate a token sequence as output. Various studies have shown that input to LLMs, which is also known as *prompt*, is critical in achieving desired results. As a result, *prompt engineering* has become an important methodology in utilizing LLMs [20]. Unlike previous algorithmic approaches, LLM-based solutions focused on devising effective prompt strategies to improve the overall performance of text-to-SQL [25, 23, 11, 6, 28].

Nevertheless, while previous LLM-based solutions have proposed several optimization techniques in different aspects of prompt construction, their performance could be further improved by taking advantage of the reasoning ability of LLMs. Since text-to-SQL is a complicated problem, it is rather challenging to finish it in one step even by human workers. Imagine the scenario where data scientists need to write a SQL query to express semantics that could also be presented with a natural language description. They first need to look at the meta-data of databases and identify the tables to be queried. At the same time, they should understand the given question, which would be achieved in a step-by-step manner. Finally, they would write the SQL query by taking both aspects into consideration. To simulate such human behaviors with LLMs, we need to take advantage of the reasoning ability of LLMs. Recently many efforts have been paid by researchers from the machine learning community to study the strategies of reasoning over LLMs. For example, Chain-of-thought (CoT) [35] is a complicated thinking process that guides LLMs towards accurate deduction of answers in a sequential manner; Least-to-most [42] divides the problem into multiple sub-questions and uses a separate prompt to solve each sub-question. Tree-of-thought [38] extends the CoT methods by enabling multiple reasoning paths as the same time. Such approaches have been successfully adopted in many NLP tasks such as question answering, information extraction and text summarization.

In this paper, we proposed a novel framework **R**easoning with **R**ethinking and **R**etrieval-augment (R3) to utilize the Chain-of-thought [35] strategy to construct multi-step prompts over LLMs for the text-to-SQL problem. This strategy was motivated by the process how human workers try to solve similar problems. R3 divided the process of prompt construction into three stages: schema linking, question decomposition and SQL generation. It first identified the relevant tables from the provided schema information based on the natural language question to avoid the noises in the prompt; Next it gave an instruction via the second prompt to decompose the natural language question into several easier sub-questions so as to help the LLM to understand the question step by step. In this stage, the refined schema information will serve as important contexts to guide the decomposition process. Finally, we construct another prompt with the obtained schema information in the first step and the set of sub-questions in the second step to ask the LLM to generate the SQL query accordingly. Another issue to be resolved in this process is the lack of contextual information related to the given question. To address this problem, we employ the Retrieval-augmented

Generation (RAG) strategy [15] to obtain contexts that are relevant to the natural language question from external documents so as to enrich the context for prompt. We conduct an extensive set of evaluation on the popular Spider dataset [39] and its three variants [10, 3, 9]. The experimental results showed that our R3 method achieved state-of-the-art performance and outperformed existing solutions by an obvious margin.

The rest of this paper is organized as following: Section 2 surveyed the related work; Section 3 introduced necessary background knowledge and presented the proposed solution; Section 4 reported the experimental results. Section 5 concluded the whole paper.

2 Related Work

There is a long stream of research on the topic of text-to-SQL from both database and NLP communities. Earlier studies [29] employed rule-based methods that first converted the natural language question into an intermediate representation and then mapped it into a SQL abstract syntax tree with heuristic rules. Later works [30, 18, 2, 26] utilized deep learning techniques to develop solutions capable of supporting cross-domain adaptation and handle complicated queries. The basic idea is to formulate text-to-SQL as a machine translation problem and then use various models with encoder-decoder architecture to solve it. With the development of pre-trained language model, fine-tuning PLM has been many data management tasks [21, 33, 31, 34, 22] including text-to-SQL. Some recent methods, such as [8, 16] further proposed sketching-based solutions to regularize the syntax of generated SQL queries via pre-defined templates.

Due to the profound capabilities of LLMs, recently LLM-based solutions have significantly outperformed previous methods and achieved promising performance. C3 [5] developed prompt strategies under zero-shot setting. Rajkumar et al. [27] conducted empirical study of text-to-SQL over different kinds of LLMs. PURPLE [28] focused on improving the schema linking step to include relevant tables in the prompt. DAIL-SQL [11] and Augment [23] studied the strategies of demonstration selection to improve the performance of few-shot in-context learning. Meta-SQL [6] aimed at improving the ranking mechanism among results returned by different prompts. CodeS [17] aimed at fine-tuning smaller open source LLMs instead of prompt engineering over larger ones. Among existing works, only DIN-SQL [25] adopted Chain-of-thought based strategies. Compared with it, our proposed framework focused on question decomposition in finer granularity and utilized RAG to boost the overall performance in each reasoning step and thus achieve much better performance.

In addition to devising solutions for the text-to-SQL problem, there are also many efforts in creating benchmarking datasets for it. Many comprehensive benchmarking tasks covered multiple application domains and involved many challenging queries, such as WikiSQL [41], Spider [39], KaggleDBQA [14] and BIRD [18]. Among them Spider is the most popular benchmarking datasets and there are also some variants of it, such as Spider-DK [10], Spider-SYN [9] and

Spider-Realistic [3]. There are also many domain-specific datasets, such as those for IMDB and Yelp [37], Advising [7], BookSQL for accounting and business [13], and FINSQL for finance [40].

3 Methodology

In this section, we introduced our proposed R3 framework for prompt based text-to-SQL. We first gave a high level overview of the framework in Section 3.1. Next we introduced the important stages for prompt of schema linking and question decomposition in Section 3.2 and Section 3.3, respectively. Finally we discussed how to employ the RAG strategy to enhance the overall framework in Section 3.4.

3.1 Overall Framework

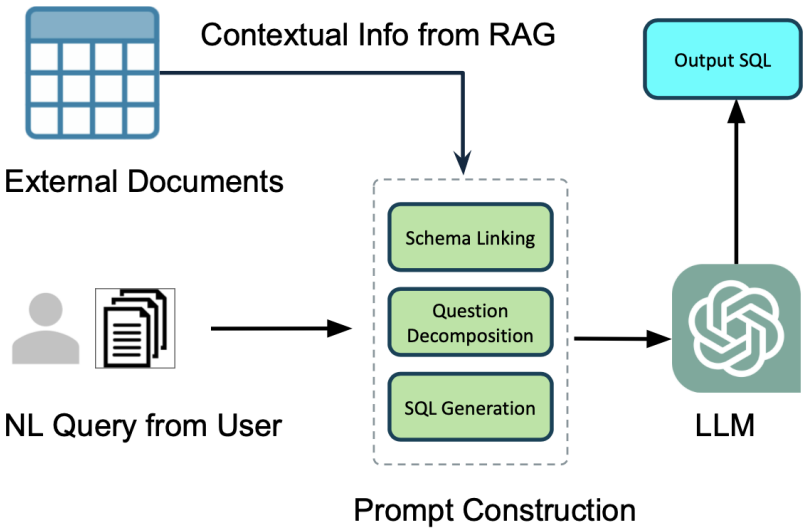

Fig. 1: The Overall Architecture of R3 Framework.

Given a natural language question X and a database D, the text-to-SQL problem aims to find an SQL query Y that expressed equivalent semantics of the question. The LLM-based solutions considered the as a sequence generation problem that employs a prompt P for the LLM M. It estimates the probability distribution over the potential SQL clause Y and generates the result in a token-

by-token manner. This generation process could be formulated as Equation 1:

$$Pr_M(Y|P,D,X) = \prod_{i=1}^{|Y|} Pr_M(Y_i|P,D,X,Y[0...i-1]) \quad (1)$$

where $Y[0...i-1]$ is the sequence generated by the model so far before step i.

In this work, we proposed a LLM based framework R3 for the text-to-SQL problem. The overall architecture is shown in Figure 1. Among them the core part is prompt construction which consists of the following three stages:

Schema Linking is the stage of finding tables that are potentially relevant to the given natural language question. It aims at filtering out irrelevant tables so as to reduce noises in the input prompt for SQL generation. We will introduce more details about it in Section 3.2.

Question Decomposition is the sage of decomposing the complex natural language question into smaller sub-questions that can answer the original question in a step-by-step manner. We will introduce more details about it in Section 3.3.

SQL Generation is the stage of asking the LLM to generate the corresponding SQL queries. The prompt consists of four sections: The *Instruction* section to describe the generation task; the *Schema* section lists the tables produced in the Schema Linking stage; The *Demonstration* section provides few-shot examples of pairs of SQL and question for in-context learning. Here we use the decomposed sub-questions to replace the original question in the demonstration example; And the *Question* section includes the list of sub-questions obtained in the Question Decomposition stage.

In addition, we also recognized that additional contextual information could help improve the quality of prompt results. To identify such information and add them into the prompts, we further employed the RAG strategy to find relevant text chunks from external resources and add them into the prompts in different stages. The details will be introduced in Section 3.4.

3.2 Schema Linking

We first look at the schema linking stage. It is an important component of text-to-SQL and has been studied by many previous studies such as [8, 28]. While these works tried to train another model to identify the tables relevant to the given question, here we reach this goal by constructing a prompt and let LLM decide the relevant tables. To this goal, we need to provide the schema of all tables as well as the natural language question. And here we regard this problem as a zero-shot in-context learning one without demonstration examples. The prompt template for the schema linking stage is shown in Figure 2. There are three sections in this template: The Instruction section describes the task of selecting the relevant tables based on the given question so as to let LLM understand the target. The Schema section lists the schema information, which includes the name and type of each column as well as the foreign and primary keys, of all tables

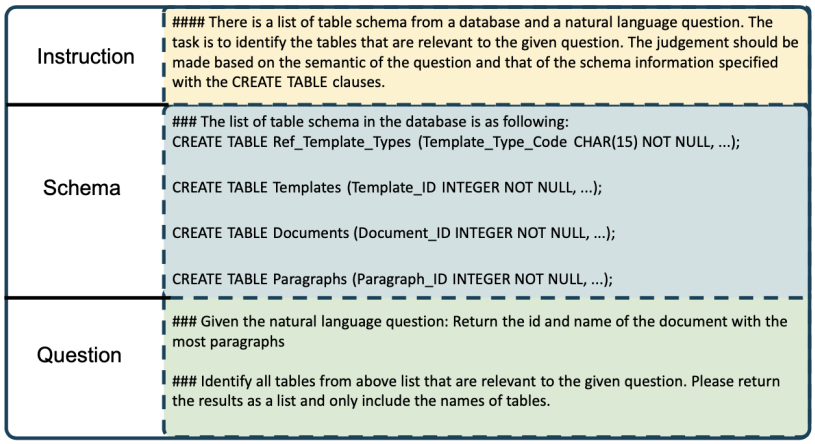

Fig. 2: The prompt template for schema linking. The DDL of each table is not fully included due to space limitation.

from the given database. We follow the practice of previous work [11] to express the schema with the Data Definition Language (DDL) starting with CREATE TABLE clauses. The Question section provides the natural language question and gives a further instruction of asking LLM to return the list of table names.

In the example in Figure 2, we select a question from dev set of Spider [39]. For this specific example, the outcome of schema linking is the list {Paragraphs, Document}. This could be deduced from the semantics of keywords in the question. Specifically, the keyword "paragraph" and "document" is syntactically similar with the table name "Paragraphs" and "Document". And the LLM could understand the semantic of the question and make a judgment that the potential query might need to rely on both tables to get the answer. Although this example seems to be a simple case, for more challenging cases such as those in the Spider variants [10, 9, 3], the task of linking the semantics between keywords and phrases in the question could still be handled by the LLMs. In some cases, there might be table names that did belong to the given list of tables in the result. To address such issues, we first use the syntactic similarity, i.e. Jaccard and edit distance, to map the results to given table names. If there is no mapping for an item in the result, we will consider it as hallucination and remove it.

3.3 Question Decomposition

Then we introduce the stage of query decomposition, which is essential to utilize the reasoning ability of LLM. It aims at decomposing a complex question into multiple simple sub-questions which can be answered in a step-by step manner. And the answer of the original question could be obtained from these sub-questions later in the SQL Generation stage. In this process, it could fully utilized the

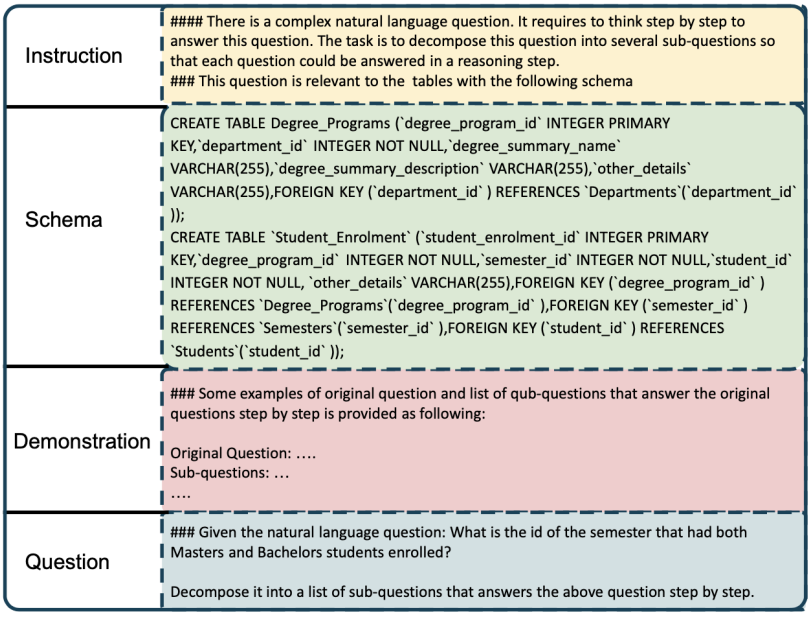

Fig. 3: The prompt template for question decomposition. The demonstration examples are omitted from this figure due to space limitation and will be introduced later.

reasoning power of LLMs to answer complex question since the detailed reasoning steps are provided by the sub-questions.

To this end, we devise the prompt template for query decomposition shown in Figure 3. It consists of the following sections: the *Instruction section* includes the essential instructions of the query decomposition task. The *Schema section* includes the schema of relevant tables, which is the output of the schema linking stage. In this stage, we consider few-shot in-context learning with examples. The reason is that query decomposition is a complicated problem and it is difficult to give a clear definition of formats in which the outcome of LLMs should be. And the demonstration examples could help provide useful information for LLM to get some senses about the output format. Thus we manually created two demonstration examples from the training set of Spider [39] following the previous work of question decomposition with LLM [36] and fixes them as the *Demonstration section* of the prompt in this stage for all test instances. Note that the questions for demonstration do not have to come from the Spider data and any resources of question decomposition could be utilized to fulfill this goal. The details of the two demonstrations are shown in Figure 4. Finally, the *Question section* presents the question to be decomposed and the guidelines to LLM. And the output from LLM for the question in Figure 3 is a list of questions as following:

– Find semesters having Master students enrolled.
– Find semesters having Bachelor students enrolled.
– Make a union of the results of semesters and formulate results.
– Select the ids of each semester in the results.

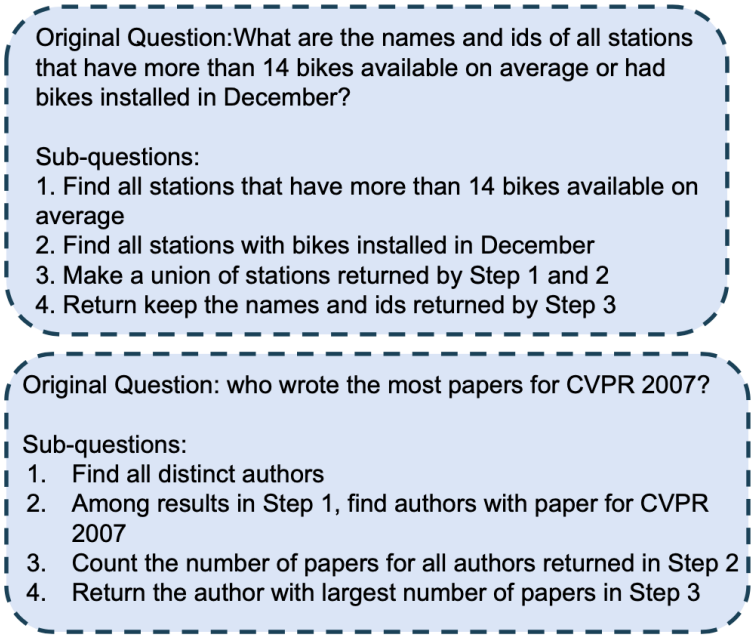

Fig. 4: Two Demonstration Examples for Question Decomposition.

We are aware that the least-to-most prompt strategy [42] also utilizes a prompt to decompose one complex question into several sub-questions. However, the least-to-most strategy requires to launch a separate prompt for each sub-question, while we just aim at obtaining the list of sub-questions as a whole and feed them into one prompt in the SQL Generation stage. Since the sub-questions of text-to-SQL is much easier than more complicated use cases introduced by [42], the list of sub-questions itself already provides sufficient insights for the question understanding and SQL generation. And it is not necessary to apply a standalone prompt for each of them, which might lead to unnecessary cost of LLM inference.

3.4 Integration of RAG Strategy

In the above in-context learning processes, a bottleneck would be that there are ambiguities in the natural language question due to the lack of contextual

information. For example, suppose we have a question "What are the open dates and years for the Apple shop in Los Angeles?". In this question, it is critical to correctly understand the term apple which refers to the brand name rather than a fruit [1]. If we also include a detailed explanation that Apple is a famous brand for electronic devices and the term "Apple shop" is usually adopted to describe the shops for that brand, it could be helpful in avoiding the ambiguity and thus the LLM could realize the semantics of the question.

To this end, we proposed to integrate the Retrieval-Augmented Generation (RAG) [15] strategy into the proposed framework to obtain additional contextual information. RAG is a popular technique that could identify small text chunks relevant to a given query. The original motivation of RAG is to help reduce the volume of contents in the input of LLM as well as improve the accuracy of retrieval based tasks. In this work, we employ this idea for a different purpose: finding relevant text chunks to question so as to enrich the context of prompt.

Our implementation of RAG is as following: The source of RAG is an external document corpus such as Wikipedia articles. We split the documents into text chunks with equal number of tokens. Then for each text chunk, we employ a pre-trained BERT [4] encoder [2] to obtain its vector embedding. After that we index the embedding vectors with vector DB related techniques. Specifically, we use the FAISS framework [3] as the index in our implementation. When we have the natural language question, we will first obtain its vector embedding using the same BERT encoder as the query. Then we retrieve the indexed vectors with the query and return the top-K similar results. The number of K is decided by the token budget of RAG, i.e. token budget divides chunk size. It is necessary to introduce the hyper-parameter of token budget to limit the overall length of the input prompt so as to save the cost over LLM. We will show more empirical results about the hyper-parameter of token budget in Section 4 later.

The results of RAG could be integrated into the prompts in all the three stages. This could be realized by adding an additional section of "Contextual Information" in the prompt templates introduced above. Specifically, we will add it under the "Instruction" section in the templates of Figure 2 and Figure 2, and add it under the "Question" section of the SQL Generation stage.

4 Experiments

4.1 Experiment Setup

Dataset We mainly conduct our experiments on the Spider dataset [39]. It consists of 200 databases from several domains and 1,056 tables in total, while there are more than 10,000 pairs of NL question and SQL (denoted as instances

[1] It cannot be easily judged from whether the word starts withe capital letter due to potential data quality issues.

[2] We use this checkpoint in our implementation: https://huggingface.co/google-bert/bert-base-uncased

[3] https://github.com/facebookresearch/faiss

in the rest of this section) in the dataset. Specifically, we conduct evaluation on its dev set that has 1,034 instances. And there are 20 databases and 81 tables in the dev set.

In addition, we also evaluate our proposed method against several previous LLM based solutions on three variants of Spider which are more challenging:

- Spider-DK [10] is a variant of Spider that requires domain specific knowledge to understand the natural language question. It is constructed based on instances from Spider-dev and has 535 instances from 10 databases.
- Spider-SYN [9] revised the Spider dataset by replacing some entities in the natural language questions with their synonyms, which makes the question understanding more difficult. It has 1.034 instances from 20 databases as same with Spider-dev.
- Spider-Realistic [3] modifies the questions in Spider-dev by removing the phrases in natural language questions that exactly matches the column names in given tables. Therefore, it could better reflect the characteristics of questions in real scenarios. It has 508 instances from 20 databases,

Although there might be many other domain specific datasets for text-to-SQL [37, 7, 13, 40], their coverage might not be as good as the Spider datasets. At the same time, the BIRD dataset [18] primarily focused on evaluating the efficiency of the generated SQL queries, which is out of the scope of this paper. Therefore, we did not use them in the experiments here due to the space limitation.

Evaluation Metrics In our experiments, we select Exact Set Match (EM) and Execution Match (EX) accuracy as the evaluation metric following the route of previous studies [11, 25]. Exact Set Match accuracy (EM), a.k.a logical form accuracy, measures the matched SQL keywords between the predicted SQL query and the corresponding ground truth. Execution Match (EX) evaluates whether the execution result matches the ground truth by executing the generated SQL query against the underlying relational database. Due to the reason that different SQL clauses might represent the equivalent semantic, the EX metric is commonly more precise than the EM one especially for LLM based solutions.

Baseline Methods On the Spider dataset, we compared the performance of the proposed R3 framework with both LLM and non-LLM solutions from previous studies. The non-LLM solutions includes PICARD [30], CatSQL [8], Graphix-T5 [19], RESDSQL [19], RASAT [26] and RYANSQL [2]. while the LLM based solutions includes DIN-SQL [25], DAIL-SQL [11], PURPLE [28], Meta-SQL [6] and CodeS [17]. For these methods, we directly cite their results from the original papers to make it consistent. On the three Spider variants dataset, we mainly compared with the LLM based solutions DIN-SQL [25], DAIL-SQL [11] and PURPLE [28] since most previous works using these datasets only reported results of the EM metric. We ran the experiments of [11] by ourselves with their source code from the authors and directly cited the results of [28].

Detailed Settings We conducted prompts over two kinds of LLMs: GPT-3.5 and GPT-4. We implemented our framework in Python and use the official APIs from OpenAI to obtain answers from LLMs. The temperature in API calls is set to 0.

In the prompt of the SQL Generation stage, we created 3 demonstration examples by manually decomposing the natural language question into a list of sub-questions. And the same set of demonstration examples is applied to all test instances in all datasets. For the implementation of RAG mechanism, we use the Wikipedia web corpus from the previous study [12] as the source document of contextual information, and we set the chunk size as 50 tokens when splitting the documents. The token budget for RAG is set as 300 by default. This choice is made according to empirical studies about the effect of token budget as shown later in this section. And we found that applying the RAG on the Schema Linking and Question Decomposition stages could always result in performance gain; while the effect of applying RAG on the SQL Generation stage is not so obvious. The detailed experimental results are omitted here due to space limitation. Thus we only apply RAG on the first two stages in all methods below which are equipped with RAG.

4.2 Main Results

Table 1: Main Results on Spider-dev. "-" means the corresponding result is not available in the original paper

Method	EM (%)	EX (%)
RYANSQL	66.4	58.2
LGESQL	75.1	34.8
PICARD	75.5	79.3
RASAT	74.7	80.5
Graphix-T5	77.1	81.0
RESDSQL	80.5	84.1
CatSQL	**80.6**	83.7
DIN-SQL	60.1	74.2
Meta-SQL	69.6	76.8
DAIL-SQL	71.9	82.4
CodeS	-	85.4
PURPLE	80.5	87.8
R3 (GPT-3.5)	68.3	82.7
R3 (GPT-4)	74.0	**88.1**

First we reported the results on Spider-dev in Table 1. We can see that R3 achieved the best performance among all methods. This is mainly due to its advantage in multi-step reasoning that lead to a better understanding of the

question. Besides, we also have the following observations: Firstly, PURPLE [6] achieved the best performance among all baseline methods. This might be due to their effective strategies in the schema linking step as well as the extra efforts in database adaption. Secondly, the previous work DIN-SQL that also employed similar idea fo chain-of-thought did not have a promising results. The reason might be that although it also tried the decompose the text-to-SQL problem into multiple steps, it failed to decompose the process of question understanding. Therefore, DIN-SQL cannot fully utilize the reasoning capacities of LLM like our R3 method did. Lastly, the performance of EX accuracy of LLM based methods are generally better than that of non-LLM ones. However, non-LLM methods have a generally better EM accuracy. The reason is that non-LLM solutions trained the decoder model by leveraging the training data, which has many instances that have similar structures with SQL clauses in the dev set. Meanwhile, LLM based methods relied on the knowledge of pre-trained models without referring to a pre-defined training data, thus it is likely to generate SQL clauses that have the equivalent semantics but not exactly the same with ones in the ground truth.

Table 2: Results on Three Variants of Spider Dataset.

Method	Spider-DK		Spider-SYN		Spider-Realistic	
	EM (%)	EX (%)	EM (%)	EX (%)	EM (%)	EX (%)
DIN-SQL	40.2	66.7	58.6	65.5	60.4	70.5
DAIL-SQL	59.8	70.2	61.5	71	68.6	76.3
PURPLE	61.7	75.3	63.3	74	**71.1**	79.9
R3 (GPT-3.5)	58.6	70.8	57.5	71.7	65.4	75.5
R3 (GPT-4)	**63.6**	**76.5**	**65.2**	**76.1**	69.8	**80.2**

Next we look at the results of three variants of Spider datasets in Table 2. We can see that R3 consistently outperformed other LLM based solutions on all datasets. Compared with the original Spider datasets, the tasks of these datasets are more challenging in the aspect of understanding the natural language questions. Our efforts of decomposing the complex question into several sub-questions could fill this gap and alleviate the problem causes by semantic understanding. Such results illustrate the potential ability to apply our solution to text-to-SQL tasks in real scenarios where question understanding is more challenging.

4.3 In-depth Analysis

We also conduct comprehensive ablation study to evaluate the effect of each component of R3. To this end, we proposed the following methods by removing the components one by one: No SL removes the prompt for schema linking and directly include schema information of all tables in the prompt; No QD removes the prompt for question decomposition by directly including the original

Table 3: Ablation Study Results using GPT-4 on Each Dataset (EX%)

Method	Spider-dev	Spider-DK	Spider-SYN	Spider-Realistic
No SL	85.8	74.2	74.9	76.2
No QD	86.3	71.3	71.1	74.6
No RAG	87.5	71.4	73.4	74.8
Full	88.1	76.5	76.1	80.2

natural language question in the final prompts. No RAG disabled the RAG mechanism and not include the contexts obtained with RAG in the prompt. And Full is the fully fledged R3 framework that is reported in Table 1 and Table 2. From the results shown in Table 3, we observe that the proposed techniques did contribute to the performance improvement. Specifically, on Spider-dev the schema linking component helps the most. At the same time, on the other three Spider variants, the RAG method provided more performance gain since the question understanding process, which might be the bottleneck on these datasets due to the challenges introduced when creating them, could benefit from the introduced contextual information.

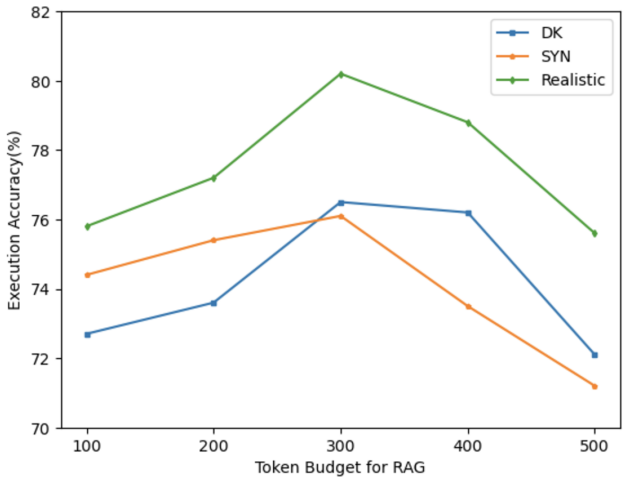

Fig. 5: The Effect of Token Budgets for RAG.

Finally, we also report the effect of token budget for RAG. As is shown in the previous results, the performance gain brought by RAG is most obvious on the three variants of Spider. Thus we exclude the evaluation on Spider-dev here. The results are shown in Figure 5. We can see that generally speaking, the overall performance becomes better. This illustrates that external resources could provide valuable contextual information for the prompt. However, when

the budget reaches 300 tokens, the improvement becomes marginal with the increasing number of tokens. And in some cases the performance might even become worse. The reason might be that when the budgets became larger, some irrelevant contents are included and thus introduced noises to the prompt. To sum up, RAG based approaches did show some promising results to the text-to-SQL problem. And it is worthy further exploring advanced RAG strategies for the text-to-SQL problem, such as using Knowledge Graph, performing retrieval on finer granularity and customizing RAG strategy for different stages, in the future work.

5 Conclusion

In this paper, we studied the problem of text-to-SQL based on prompt engineering over LLMs. We proposed R3, a novel framework to employ chain-of-thought strategy to decompose the text-to-SQL problem into three different stages: schema linking, question decomposition and SQL generation and developed effective prompt strategies for each stage. We further integrated RAG techniques into this framework to retrieve essential contextual information so as to enrich the content of the prompt. Experimental results on several public benchmarking datasets showed the superiority of our proposed solutions. For the future work, we will aim at developing more advanced RAG techniques based on richer external resources such as Knowledge Graph and studying how to reduce the number of tokens in the input of LLM to save the overall cost. Besides, it is also a promising direction to study how to decompose the question as well as design the reasoning steps based on domain specific knowledge. In this way, the proposed techniques could be further applied to real world scenarios.

Acknowledgment

This work was supported by the National Science Foundation of China (Grant No. 62232009).

References

1. Chen, M., et al.: Evaluating large language models trained on code. CoRR **abs/2107.03374** (2021)
2. Choi, D., Shin, M., Kim, E., Shin, D.R.: RYANSQL: recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. Comput. Linguistics **47**(2), 309–332 (2021)
3. Deng, X., Awadallah, A.H., Meek, C., Polozov, O., Sun, H., Richardson, M.: Structure-grounded pretraining for text-to-sql. In: NAACL-HLT. pp. 1337–1350 (2021)
4. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT. pp. 4171–4186 (2019)

5. Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Chen, L., Lin, J., Lou, D.: C3: zero-shot text-to-sql with chatgpt. CoRR **abs/2307.07306** (2023)
6. Fan, Y., He, Z., Ren, T., Huang, C., Jing, Y., Zhang, K., Wang, X.S.: Metasql: A generate-then-rank framework for natural language to SQL translation. In: ICDE. pp. 1765–1778 (2024)
7. Finegan-Dollak, C., Kummerfeld, J.K., Zhang, L., Ramanathan, K., Sadasivam, S., Zhang, R., Radev, D.R.: Improving text-to-sql evaluation methodology. In: ACL. pp. 351–360 (2018)
8. Fu, H., Liu, C., Wu, B., Li, F., Tan, J., Sun, J.: Catsql: Towards real world natural language to SQL applications. Proc. VLDB Endow. **16**(6), 1534–1547 (2023)
9. Gan, Y., Chen, X., Huang, Q., Purver, M., Woodward, J.R., Xie, J., Huang, P.: Towards robustness of text-to-sql models against synonym substitution. In: ACL/IJCNLP. pp. 2505–2515 (2021)
10. Gan, Y., Chen, X., Purver, M.: Exploring underexplored limitations of cross-domain text-to-sql generalization. In: EMNLP. pp. 8926–8931 (2021)
11. Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J.: Text-to-sql empowered by large language models: A benchmark evaluation. Proc. VLDB Endow. **17**(5), 1132–1145 (2024)
12. Karpukhin, V., Oguz, B., Min, S., Lewis, P.S.H., Wu, L., Edunov, S., Chen, D., Yih, W.: Dense passage retrieval for open-domain question answering. In: EMNLP. pp. 6769–6781 (2020)
13. Kumar, R., Dibbu, A.R., Harsola, S., Subrahmaniam, V., Modi, A.: Booksql: A large scale text-to-sql dataset for accounting domain. In: NAACL. pp. 497–516 (2024)
14. Lee, C., Polozov, O., Richardson, M.: Kaggledbqa: Realistic evaluation of text-to-sql parsers. In: ACL/IJCNLP. pp. 2261–2273 (2021)
15. Lewis, P.S.H., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: NeurIPS (2020)
16. Li, H., Zhang, J., Li, C., Chen, H.: RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql. In: AAAI. pp. 13067–13075 (2023)
17. Li, H., Zhang, J., Liu, H., Fan, J., Zhang, X., Zhu, J., Wei, R., Pan, H., Li, C., Chen, H.: Codes: Towards building open-source language models for text-to-sql. Proc. ACM Manag. Data **2**(3), 127 (2024)
18. Li, J., et al.: Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In: NeurIPS (2023)
19. Li, J., Hui, B., Cheng, R., Qin, B., Ma, C., Huo, N., Huang, F., Du, W., Si, L., Li, Y.: Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In: AAAI. pp. 13076–13084 (2023)
20. 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), 195:1–195:35 (2023)
21. Lu, J., Lin, C., Wang, J., Li, C.: Synergy of database techniques and machine learning models for string similarity search and join. In: CIKM. pp. 2975–2976 (2019)
22. Miao, Z., Wang, J.: Watchog: A light-weight contrastive learning based framework for column annotation. Proc. ACM Manag. Data **1**(4), 272:1–272:24 (2023)
23. Nan, L., Zhao, Y., Zou, W., Ri, N., Tae, J., Zhang, E., Cohan, A., Radev, D.: Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies. In: Findings of the Association for Computational Linguistics: EMNLP. pp. 14935–14956 (2023)

24. OpenAI: GPT-4 technical report. CoRR **abs/2303.08774** (2023)
25. Pourreza, M., Rafiei, D.: DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In: NeurIPS (2023)
26. Qi, J., Tang, J., He, Z., Wan, X., Cheng, Y., Zhou, C., Wang, X., Zhang, Q., Lin, Z.: RASAT: integrating relational structures into pretrained seq2seq model for text-to-sql. In: EMNLP. pp. 3215–3229 (2022)
27. Rajkumar, N., Li, R., Bahdanau, D.: Evaluating the text-to-sql capabilities of large language models. CoRR **abs/2204.00498** (2022)
28. Ren, T., Fan, Y., He, Z., Huang, R., Dai, J., Huang, C., Jing, Y., Zhang, K., Yang, Y., Wang, X.S.: PURPLE: making a large language model a better SQL writer. In: ICDE. pp. 15–28 (2024)
29. Saha, D., Floratou, A., Sankaranarayanan, K., Minhas, U.F., Mittal, A.R., Özcan, F.: ATHENA: an ontology-driven system for natural language querying over relational data stores. Proc. VLDB Endow. **9**(12), 1209–1220 (2016)
30. Scholak, T., Schucher, N., Bahdanau, D.: PICARD: parsing incrementally for constrained auto-regressive decoding from language models. In: EMNLP. pp. 9895–9901 (2021)
31. Shahbazi, N., Wang, J., Miao, Z., Bhutani, N.: Fairness-aware data preparation for entity matching. In: ICDE. pp. 3476–3489 (2024)
32. Touvron, H., et al.: Llama: Open and efficient foundation language models. CoRR **abs/2302.13971** (2023)
33. Wang, J., Li, Y.: Minun: evaluating counterfactual explanations for entity matching. In: DEEM '22. pp. 7:1–7:11. ACM (2022)
34. Wang, J., Li, Y., Hirota, W., Kandogan, E.: Machop: an end-to-end generalized entity matching framework. In: aiDM '22. pp. 2:1–2:10 (2022)
35. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: NeurIPS (2022)
36. Wolfson, T., Deutch, D., Berant, J.: Weakly supervised text-to-sql parsing through question decomposition. In: Findings of the Association for Computational Linguistics: NAACL. pp. 2528–2542 (2022)
37. Yaghmazadeh, N., Wang, Y., Dillig, I., Dillig, T.: Sqlizer: query synthesis from natural language. Proc. ACM Program. Lang. **1**(OOPSLA), 63:1–63:26 (2017)
38. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with large language models. In: NeurIPS (2023)
39. Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., Radev, D.R.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: EMNLP. pp. 3911–3921 (2018)
40. Zhang, C., Mao, Y., Fan, Y., Mi, Y., Gao, Y., Chen, L., Lou, D., Lin, J.: Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. In: Companion of SIGMOD/PODS. pp. 93–105. ACM (2024)
41. Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR **abs/1709.00103** (2017)
42. Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q.V., Chi, E.H.: Least-to-most prompting enables complex reasoning in large language models. In: ICLR (2023)