Gen-SQL: Efficient Text-to-SQL By Bridging Natural Language Question And Database Schema With Pseudo-Schema

Jie Shi[1], Bo Xu[2]*, Jiaqing Liang[3]*, Yanghua Xiao[1], Jia Chen[1], Chenhao Xie[4], Peng Wang[1], Wei Wang[1]

[1]Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
[2]School of Computer Science and Technology, Donghua University
[3]School of Data Science, Fudan University [4]SenseDeal Intelligent Technology Co., Ltd.
jshi22@m.fudan.edu.cn, xubo@dhu.edu.cn, xiechenhao@sensedeal.ai,
{liangjiaqing,shawyh,jiachen,pengwang5,weiwang1}@fudan.edu.cn

Abstract

With the prevalence of Large Language Models (LLMs), recent studies have shifted paradigms and leveraged LLMs to tackle the challenging task of Text-to-SQL. Because of the complexity of real world databases, previous works adopt the retrieve-then-generate framework to retrieve relevant database schema and then to generate the SQL query. However, efficient embedding-based retriever suffers from lower retrieval accuracy, and more accurate LLM-based retriever is far more expensive to use, which hinders their applicability for broader applications. To overcome this issue, this paper proposes Gen-SQL, a novel generate-ground-regenerate framework, where we exploit prior knowledge from the LLM to enhance embedding-based retriever and reduce cost. Experiments on several datasets are conducted to demonstrate the effectiveness and scalability of our proposed method. We release our code and data at https://github.com/jieshi10/gensql.

1 Introduction

Text-to-SQL aims to generate a SQL query that answers a natural language question given a relational database schema, which is an essential but challenging task for automating management and simplifying data access of databases (Qin et al., 2022). Nowadays, Large Language Models (LLMs; Brown et al., 2020; Chen et al., 2021a; Touvron et al., 2023b) have shown strong reasoning and generalization ability (Wei et al., 2022; Yao et al., 2023) and have been proven successful on a variety of tasks. As for Text-to-SQL, recent works (Wang et al., 2024; Pourreza and Rafiei, 2023; Liu et al., 2023; Rajkumar et al., 2022) have achieved promising results by designing sophisticated prompting techniques for LLMs, and they leverage a retrieve-then-generate framework, where database schema relevant to the question is retrieved first and the LLM is then prompted to generate the SQL query based on the retrieved schema.[1]

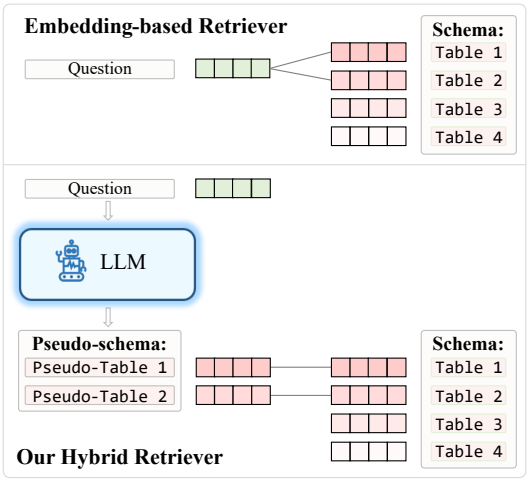


Figure 1: Bridging semantic gap with generated pseudo-schema. The embedding vector of the natural language question is different from those of the database tables. So Gen-SQL leverages the LLM to generate pseudo-tables, whose embedding vectors are more similar.

However, the retrieval step in previous works has limited applicability. Open source platforms such as DB-GPT[2] utilize embedding-based retriever which can efficiently find top-K relevant tables (where K is a predefined constant). But embedding-based retriever suffers from lower retrieval accuracy. As shown in Figure 1, this is due to the semantic gap between natural language question and database schema (Baik et al., 2019). Correctly building the connection between the question and the database schema is challenging, as illustrated in Figure 2. Failure to retrieve the schema will result in suboptimal generation performance. For better retrieval performance, the most advanced Text-to-SQL method (Wang et al., 2024; Pourreza

*Corresponding authors.

[1]The retrieval step is traditionally known as *schema linking* in the literature.

[2]https://github.com/eosphoros-ai/DB-GPT

Proceedings of the 31st International Conference on Computational Linguistics, pages 3794–3807
January 19–24, 2025.

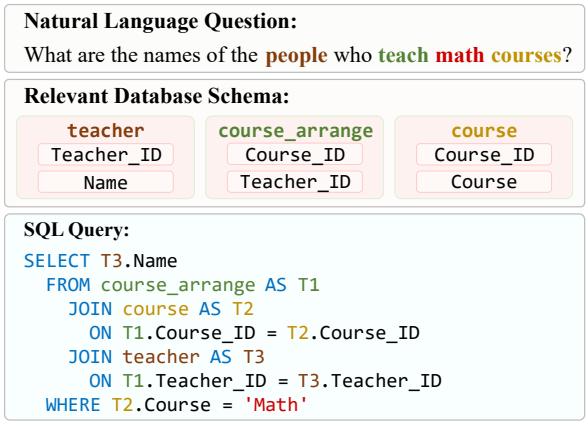


Figure 2: An example showing the semantic gap between natural language question and database schema. The question mentions two entities (*i.e.*, **people** and **courses**) and their relation (*i.e.*, **teach**). But it is still difficult to map them to correct tables (*i.e.*, **people**→`teacher`, **courses**→`course`, **teach**→`course_arrange`).

and Rafiei, 2023) relies on LLM-based retriever to search for fine-grained database schema where irrelevant columns in each retrieved table are further pruned. But scanning with LLMs will incur high cost and high latency.

To overcome these limitations, this paper proposes to make appropriate use of prior knowledge from the LLM to bridge the gap between question and schema for embedding-based retriever. The proposed generate-ground-regenerate framework for Text-to-SQL is named Gen-SQL. As shown in Figure 3, we first instruct the LLM to come up with a pseudo-schema depicting the table structure solely based on the question without providing the schema. This imaginary pseudo-schema is generated according to the prior knowledge obtained during the pre-training stage, and serves as an intermediate modality to bridge the gap between natural language and database schema. Based on the pseudo-schema, we leverage an embedding-based retriever to accurately retrieve an adaptive number of tables and prune irrelevant columns so that the pseudo-schema can be grounded in actual database schema. Meanwhile, the LLM indirectly augments the embedding-based retriever without significantly impacting overall latency and cost. Finally, we prompt the LLM with the grounded schema to regenerate the SQL query. Furthermore, we propose iterative refinement to automatically refine the generated SQL queries.

The contributions of this paper are summarized as follows:

- This paper studies the semantic gap between natural language question and database schema during schema retrieval in Text-to-SQL. We propose to leverage pseudo-schema generated by the LLM to bridge the semantic gap for more accurate retrieval.
- We propose Gen-SQL, a generate-ground-regenerate framework for Text-to-SQL. Compared with previous methods, Gen-SQL is more efficient and less expensive due to its hybrid retriever.
- In the experiments, we show that Gen-SQL outperforms the existing baselines on public benchmarks. We also validate Gen-SQL on two curated datasets featuring databases with a massive number of tables to demonstrate its scalability.

2 Preliminaries for Text-to-SQL

2.1 Problem Formulation

The input of the Text-to-SQL task is a natural language question q and a database schema $\mathcal{S} = \{s_1, \cdots, s_N\}$, where s_i is the i-th table and N indicates the number of tables in the database. For table s_i, its column collection is denoted by $\mathcal{C}_i = \{c_{i,1}, \cdots, c_{i,N_i}\}$, where $c_{i,j}$ is the j-th column and N_i is the number of columns. The output of the Text-to-SQL task is a SQL query $\hat{y}$ which corresponds to the question q.

2.2 Retrieve-then-generate Framework

The retrieve-then-generate framework prevails in many recent Text-to-SQL methods (Wang et al., 2024; Pourreza and Rafiei, 2023, 2024). It aims to help LLMs understand connections between the question and the database schema. A retriever retrieves the schema $\hat{S} = \{\hat{s}_1, ..., \hat{s}_K\} \subseteq \mathcal{S}$ relevant to the question q. Let $\hat{C}_i = \left\{\hat{c}_{i,1}, \cdots, \hat{c}_{i,\hat{N}_i}\right\}$ be the column collection for table $\hat{s}_i$, where $\hat{N}_i$ is the number of columns. The LLM is prompted with the retrieved schema $\hat{S}$ to generate the SQL query $\hat{y}$ during the generation step:

$$\hat{y} = \text{SQL-Writer}\left(\hat{S}, q\right). \quad (1)$$

The number of tables K varies for different types of retrievers.

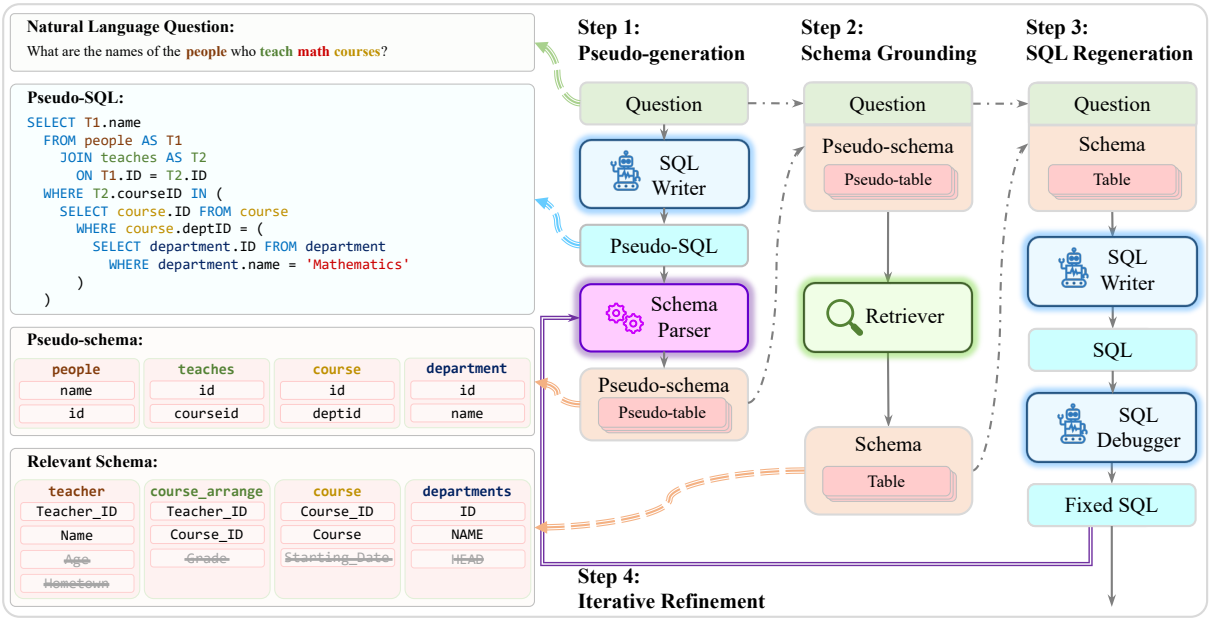


Figure 3: Compared with the retrieve-then-generate framework which ignores table structures during retrieval, Gen-SQL first leverages the LLM to generate a pseudo-schema, then performs schema grounding, and finally regenerates the SQL query. The proposed approach can be extended to an iterative framework as indicated by the **purple** arrow.

For embedding-based retriever, K is typically a predefined constant. The retriever retrieves top-K tables which are most relevant to the question q by calculating cosine similarity score $\cos(q, s_i)$ between the question q and each table $s_i \in \mathcal{S}$, and finding a subset of tables $\hat{S} = \{\hat{s}_1, ..., \hat{s}_K\} \subseteq \mathcal{S}$ with top-K similarity scores.

For LLM-based retriever, K is determined adaptively. The LLM is prompted to scan the complete schema and search for relevant tables and columns based on its own knowledge. Although this kind of retriever is more accurate and flexible, it will cause extra cost. When the schema is large, the cost will increase accordingly.

3 Proposed Approach: Gen-SQL

In standard retrieve-then-generate framework, the embedding-based retriever does not take schema into account during the retrieval step, and the LLM-based retriever has higher cost. We propose to tackle these issues by augmenting the embedding-based retriever with schema information from the LLM. Specifically, we exploit prior knowledge from the LLM to first generate a pseudo-schema (§3.1) which is later grounded in actual database schema (§3.2). The LLM then regenerates the SQL query based on the grounded schema (§3.3). The proposed generate-ground-regenerate framework for Text-to-SQL is named Gen-SQL, as shown in Figure 3.

We further extend this framework into an iterative one (§3.4), where the generated SQL query can be automatically refined at each iteration.

3.1 Step 1: Pseudo-generation

This step aims to generate a pseudo-schema $\bar{S} = \{\bar{s}_1, \cdots, \bar{s}_n\}$ related to the question q, where n is the number of pseudo-tables. Let $\bar{C}_i = \{\bar{c}_{i,1}, \cdots, \bar{c}_{i,\bar{n}_i}\}$ be the column collection for pseudo-table $\bar{s}_i$, where $\bar{n}_i$ is the number of columns in pseudo-table $\bar{s}_i$.

Due to the complexity of the `CREATE TABLE` statements, we do not prompt the LLM to directly write them. Instead, we instruct the LLM to generate a pseudo-SQL $\bar{y}$ without specifying the schema:

$$\bar{y} = \text{SQL-Writer}(\emptyset, q). \quad (2)$$

Because we do not prompt the LLM with full schema, the cost will not increase with respect to the schema size. Afterwards, the schema parser module analyzes the pseudo-SQL $\bar{y}$ by converting it to Abstract Syntax Tree (AST), based on which the schema parser composes the pseudo-schema $\bar{S}$. Please refer to Appendix A for detailed explanations for the schema parser module. Note that the pseudo-schema $\bar{S}$ containing information about the

table names ($\bar{s}_i$), column names ($\bar{c}_{i,j}$), and the numbers (n, $\bar{n}_i$) can be inferred from the pseudo-SQL $\bar{y}$.

3.2 Step 2: Schema Grounding

Schema grounding consists of two stages. It makes use of both the question q and the pseudo-schema $\bar{S} = \{\bar{s}_1, \cdots, \bar{s}_n\}$ to first retrieve top-n tables (where n is the number of pseudo-tables), and then eliminate irrelevant columns in each table.

Specifically, the question q is enriched with the pseudo-schema $\bar{S}$. We use the string concatenation $q^s = [q; \bar{s}_1; \cdots; \bar{s}_n]$ to compute the cosine similarity score $\cos(q^s, s_i)$ ($s_i \in \mathcal{S}$), and obtain a set of n tables $\hat{S} = \{\hat{s}_1, \cdots, \hat{s}_n\} \subseteq \mathcal{S}$ that are the most relevant. Let $\hat{C}_i = \{\hat{c}_{i,1}, \cdots, \hat{c}_{i,\hat{n}_i}\}$ be the column collection for table $\hat{s}_i$. The columns are sorted based on their similarities with the enriched question q^s, *i.e.*, $\cos(q^s, \hat{c}_{i,j}) > \cos(q^s, \hat{c}_{i,j'})$ for $j < j'$. We only keep the first $\hat{n}'_i$ columns for table $\hat{s}_i$:

$$\hat{n}'_i = \begin{cases} \bar{n}_j, & \exists j : \bar{s}_j = \hat{s}_i \\ \max_j \bar{n}_j, & \text{otherwise} \end{cases}. \quad (3)$$

If table $\hat{s}_i$ is mentioned in the pseudo-schema, we keep the same number of columns as that in the pseudo-schema. Otherwise, we keep the largest number of columns required in all pseudo-tables. The retrieved schema $\hat{S}$ is the grounded schema.

It is worth mentioning that the pseudo-schema $\bar{S}$ enhances the grounding step from two perspectives. First, it enriches the query context with table structures, minimizing the gap between the natural language question and the database schema. Second, it enables fine-grained retrieval by providing the number of tables to retrieve and the number of columns needed in each table.

In practice, we will keep $\lambda_1 n$ tables and $\lambda_2 \hat{n}'_i$ columns for table $\hat{s}_i$ because the retriever is not perfect.

3.3 Step 3: SQL Regeneration

This step regenerates the SQL query $\hat{y}$ based on the grounded schema $\hat{S}$. This regeneration step is similar to the generation step in standard retrieve-then-generate framework. In addition to the SQL writer module that generates the SQL query based on the grounded schema and the question, we include a SQL debugger module that fixes SQL queries failing to execute on the database.

SQL Writer. The SQL writer module takes the grounded schema $\hat{S}$ and the question q as input and generates a SQL query $\hat{y}'$ using the LLM:

$$\hat{y}' = \text{SQL-Writer}\left(\hat{S}, q\right). \quad (4)$$

In our implementation, we select 8 in-context examples from the training set based on question similarity (Li et al., 2024). The schema $\hat{S}$ is transformed into `CREATE TABLE` statements. These `CREATE TABLE` statements contain table names, column names, primary key constraints, and foreign key constraints (Gao et al., 2024). We also randomly sample one cell value for each column, so that the LLM can know the value format of each column. We leave implementing an efficient value retriever as our future work.

The detailed prompt is given in Appendix C.1.

SQL Debugger. The SQL debugger module takes the grounded schema $\hat{S}$, the question q, and the SQL query $\hat{y}'$ generated by the SQL writer module as input and produces a fixed SQL query $\hat{y}$.

Specifically, the SQL debugger executes the SQL query $\hat{y}'$ on the database. If it is successfully executed, then the SQL debugger takes no further action. Otherwise, the SQL debugger instructs the LLM to fix the SQL query $\hat{y}'$ based on the error feedback $\mathcal{E}$ from the database:

$$\hat{y} = \text{SQL-Debugger}\left(\hat{S}, q, \hat{y}', \mathcal{E}\right). \quad (5)$$

The detailed prompt is given in Appendix C.2.

3.4 Step 4: Iterative Refinement

By taking the initial SQL query $\hat{y}_1 = \hat{y}$ as input of the schema parser, the proposed approach can be made iterative.

Formally, let $\hat{y}_t$ be the SQL query generated at the t-th iteration, where $t = 1, 2, \cdots, (L-1)$ and L is the maximum number of iterations. At the $(t+1)$-th iteration, the schema parser parses the SQL query $\hat{y}_t$ and produces a pseudo-schema $\bar{S}_t$, based on which the retriever retrieves the relevant schema $\hat{S}_t$. Afterwards, the LLMs are prompted with the schema $\hat{S}_t$ and generate a new SQL query $\hat{y}_{t+1}$. The iteration stops when either the schema at the next iteration remains the same as that at the current iteration, *i.e.*, $\hat{S}_{t+1} = \hat{S}_t$, or the maximum number of iterations is reached, *i.e.*, $t + 1 = L$. In the experiments, we set $L = 5$.

Mathematically, the objective of our task is to maximize the probability of generating the SQL

	Min	Max	Avg
Spider	2	11	4
Spider-M	51	73	59
BIRD	3	13	7
BIRD-M	78	95	89

Table 1: Dataset statistics showing minimum, maximum, and average number of tables for each database.

Method	Iterative	Retrieval	Debugging
DIN-SQL	✗	✓	✓
MAC-SQL	✗	✓	✓
DAIL-SQL	✗	✗	✗
Gen-SQL	✓	✓	✓

Table 2: Comparisons between different methods. A method is "iterative" if the previously generated SQL query will serve as hint in the next iteration. "Retrieval" indicates that the method will retrieve relevant schema before SQL generation. "Debugging" suggests that the method will make self-corrections to the generated SQL.

query $\hat{y}$ by finding an optimal subset of tables $\hat{S} \subseteq \mathcal{S}$ given $\mathcal{S}$ and q:

$$\arg\max_{\hat{y},\hat{S}} p\left(\hat{y} \mid \hat{S}, q\right). \tag{6}$$

Directly optimizing this objective is intractable, so we iteratively maximize this probability in two steps. At the t-th iteration, suppose the generated SQL query is $\hat{y}_t$. At the $(t+1)$-th iteration, we first estimate $\hat{S}_t$ based on $\hat{y}_t$ to retrieve the most relevant schema. By keeping $\hat{S}_t$ fixed, we can then maximize the probability and find $\hat{y}_{t+1}$ using the decoding algorithms of LLMs:

$$\arg\max_{\hat{y}_{t+1}} p\left(\hat{y}_{t+1} \mid \hat{S}_t, q\right). \tag{7}$$

The above procedure is analogous to the expectation-maximization algorithm, where estimating $\hat{S}_t$ is the expectation (E) step and decoding $\hat{y}_{t+1}$ is the maximization (M) step. So we can interpret iterative refinement as an optimization technique.

4 Experiments

We conduct all the experiments on a server with 1TB RAM and 8 NVIDIA A40 GPUs.

4.1 Experiment Settings

4.1.1 Datasets

We conduct the experiments on two public Text-to-SQL benchmarks and two curated datasets whose databases consisting of much larger numbers of tables.

To demonstrate the effectiveness of Gen-SQL, we evaluate the proposed method on two widely used benchmarks, *i.e.*, Spider (Yu et al., 2018) and BIRD (Li et al., 2023b). We report results on the development set of each benchmark.

To illustrate the scalability of Gen-SQL, we study Text-to-SQL on complex databases with massive numbers of tables. We construct two datasets by merging databases from Spider (Yu et al., 2018) and BIRD (Li et al., 2023b). The resulting datasets are named Spider-M and BIRD-M. Note that only databases are merged. The questions and gold SQL queries in Spider-M and BIRD-M are exactly the same as those in Spider and BIRD. Details of the construction process can be found in Appendix B. Table 1 shows the statistics of the original and the resulting datasets.

4.1.2 Models

Retriever. The state-of-the-art text embedding model `bge-large-en-v1.5` (Xiao et al., 2023) is used as the embedding-based retriever.

LLMs. For open source LLM, we use the Llama-3-70B model.[3] As indicated in previous work (Gao et al., 2024), instruction fine-tuned variant of this model is used due to its superior performance. For proprietary LLM, we use the latest GPT-4 model `gpt-4o-2024-08-06` (OpenAI, 2024). Both Llama (Touvron et al., 2023b,a) and GPT-4 are general domain LLMs.

4.1.3 Implementation Details

Our code is based on the PyTorch (Paszke et al., 2019) version of the Transformers (Wolf et al., 2020) library.

Generation Configuration. Greedy decoding is used across all experiments, and the maximum number of tokens to generate for each SQL query is limited to 256.

Model Serving. Llama-3-70B is deployed on 4 GPUs using vLLM (Kwon et al., 2023) for optimal inference speed.

4.2 Baselines

We compare Gen-SQL with three state-of-the-art Text-to-SQL methods: DIN-SQL (Pourreza and

[3] https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct

Method (%)	Spider		Spider-M		BIRD		BIRD-M	
	EX	EM	EX	EM	EX	VES	EX	VES
GPT-4								
DIN-SQL ♠	74.2	60.1	-	-	50.7	58.8	-	-
MAC-SQL	81.3	45.4	-	-	57.9	62.0	-	-
DAIL-SQL ♠	82.4	**71.9**	-	-	-	-	-	-
Gen-SQL	**85.6**	64.3	-	-	**59.8**	**62.8**	-	-
GPT-3.5-Turbo								
MAC-SQL ♠	**80.6**	-	-	-	**50.6**	**61.3**	-	-
DAIL-SQL ♠	78.1	**66.7**	-	-	-	-	-	-
Llama-3-70B								
DIN-SQL	78.7	54.4	8.3	7.1	34.3	34.4	3.4	3.5
MAC-SQL	77.0	41.7	0.9	0.6	46.4	49.7	8.2	8.2
DAIL-SQL	76.5	58.9	67.0	52.7	46.4	46.4	31.9	32.3
Gen-SQL	**82.8**	**66.0**	**81.1**	**66.2**	**53.2**	**54.0**	**48.6**	**49.1**

Table 3: Performance of different methods. (♠: Results are from their original papers. **Bold**: the best within each LLM. Underlined: the second best within each LLM.)

(%) Row	Iterative	Retrieval	Debugging	Spider		Spider-M		BIRD		BIRD-M	
				EX	EM	EX	EM	EX	VES	EX	VES
1	✓	✓	✓	**82.8**	**66.0**	**81.1**	**66.2**	**53.2**	**54.0**	**48.6**	**49.1**
2	✓	✓	✗	82.1	65.5	79.7	65.1	49.0	49.4	45.1	45.1
3	✗	✓	✓	82.4	64.9	80.1	63.0	52.4	53.1	46.6	46.6
4	✗	✓	✗	82.0	64.6	79.4	63.5	47.5	47.7	41.5	41.4
5	✗	✗	✓	80.9	63.5	71.6	54.5	51.6	51.9	31.6	32.3
6	✗	✗	✗	79.8	63.4	66.9	51.5	46.8	47.2	25.4	25.9
7	✗	Top-5	✓	81.4	63.8	79.6	62.6	50.6	52.6	42.0	42.8
8	✗	Top-10	✓	81.6	64.4	79.0	62.2	51.0	51.6	43.0	44.0

Table 4: Ablation study based on Llama-3-70B. (**Bold**: the best.)

Rafiei, 2023), MAC-SQL (Wang et al., 2024), and DAIL-SQL (Gao et al., 2024).

DIN-SQL (Pourreza and Rafiei, 2023) and MAC-SQL (Wang et al., 2024) fully rely on LLMs to perform schema retrieval, SQL generation, and SQL debugging.

DAIL-SQL (Gao et al., 2024) mainly focuses on prompt organization and few-shot demonstration selection. It does not perform schema retrieval and prompts LLMs with full database schema.

The differences between Gen-SQL and the baselines are summarized in Table 2. All baselines are implemented based on their official code. Since the original implementations do not take large database schemas into account, we additionally perform truncation to prompts exceeding the context limit of LLM so as to avoid errors. Due to budget limitation, we conduct experiments using GPT-4 only on public benchmarks.

4.3 Metrics

For Spider and Spider-M, we report Execution Accuracy (EX) and Exact Matching Accuracy (EM) (Yu et al., 2018; Li et al., 2023a). For BIRD and BIRD-M, we report Execution Accuracy (EX) and Valid Efficiency Score (VES) (Li et al., 2023b). EX is defined as the result matching accuracy of executing the generated SQL query against executing the gold SQL query on the given database. EM measures whether the decomposed SQL components of the generated query match those of the gold query. VES measures the execution efficiency of the generated query against the gold query.

4.4 Main Results

Table 3 shows the performance of different methods.

Effectiveness. On Llama-3-70B, Gen-SQL outperforms the other baselines on public benchmarks like Spider and BIRD, and achieves an EX of 82.8% and 53.2%, which is comparable to the results achieved on proprietary LLM like GPT-3.5-Turbo. On GPT-4, the performance of Gen-SQL is also compelling.

Scalability. It can be observed that the performance of available methods significantly decreases on Spider-M and BIRD-M. For example, from BIRD to BIRD-M, the performance of DIN-SQL and MAC-SQL drops to below 10% EX. This is because these methods rely on the instruction-following ability of LLMs. When the schema con-

Method	Spider			Spider-M		
	EX (%)	# Tokens	Latency (s)	EX (%)	# Tokens	Latency (s)
DIN-SQL	78.7	8,953	35.1	8.3	7,642	33.2
MAC-SQL	77.0	2,239	18.8	0.9	15,154	34.4
DAIL-SQL	76.5	**767**	**4.5**	67.0	4,155	**7.1**
Gen-SQL	**82.8**	1,473	6.1	**81.1**	**2,348**	8.4

Table 5: Efficiency of different methods based on Llama-3-70B. (**Bold**: the best. Underlined: the second best.)

text is long, the LLM fails to correctly follow the given instructions. Surprisingly, the EX of DAIL-SQL only drops by 14.5% on BIRD-M. This is because DAIL-SQL relies on the in-context learning ability of LLMs, which appears to be more robust than the instruction-following ability. In comparison, the deterioration of Gen-SQL on BIRD-M in terms of EX is marginal (less than 5%), which proves that the proposed method scales well on larger databases.

4.5 Ablation Study

This section examines the effectiveness of each component in the proposed method. Table 4 shows the results. We also include the variants of retrieving top-K tables (rows 7 and 8) using embedding-based retriever to justify our retrieving strategy.

Overall Framework. The results show that removing any component leads to inferior performance. Specifically, without self-debugging (row 2), the EX of Gen-SQL drops by 3.5% on BIRD-M. Without the iterative framework (row 3), the EM drops by 3.2% on Spider-M. When we further remove schema retrieval (row 5), the EX decreases by 15.0% on BIRD-M compared with non-iterative Gen-SQL (row 3). These prove the effectiveness of the proposed framework.

Generation Prompt. The generation prompts used in our implementation are comparable or even superior to the baselines.

For instance, the results from row 6 are directly comparable to those of DAIL-SQL, since both methods are non-iterative and generate SQL without schema retrieval or self-debugging. Their performance is similar on Spider and BIRD. However, our method slightly underperforms DAIL-SQL on Spider-M and BIRD-M. The major difference between our method and DAIL-SQL is that our method takes database content into consideration. It appears that database content has little impact on smaller databases, but it has negative influence on larger ones due to increased prompt lengths.

(%)	P	R	F1	EX
Embedding-based Retriever				
Top-5	**30.8**	80.0	**44.5**	42.0
Top-10	17.3	**89.8**	29.0	**43.0**
Hybrid Retriever				
Gen-SQL	23.1	**93.6**	37.1	**48.6**
Non-iterative	**25.1**	90.1	**39.2**	46.6

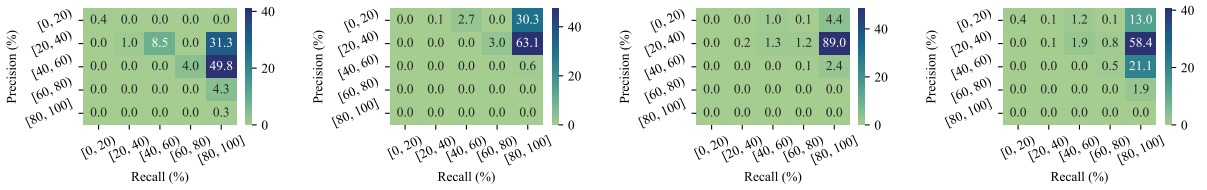
Table 6: Table retrieval and Text-to-SQL performance on BIRD-M. Llama-3-70B is used as the backbone. (**Bold**: the best within each retriever.)

Similarly, the results from row 3 are almost comparable to those of DIN-SQL and MAC-SQL, because all methods are non-iterative and perform schema retrieval and self-debugging. Our method outperforms the other baselines by a large margin on all datasets, which validates the prompting strategy of Gen-SQL.

Retrieving Strategy. The results also demonstrate the superiority of the proposed retrieving strategy. Comparing the results from row 3 with those from rows 7 and 8, the retrieving strategy of Gen-SQL leads to better performance. Specifically, on larger and more complex dataset like BIRD-M, the improvement is significant (↑3.6% EX).

4.6 Analysis

Efficiency. We measure the efficiency using average token consumption and latency for each sample. The results are presented in Table 5. DAIL-SQL has the lowest latency because it will only invoke the LLM once for a single question. In comparison, DIN-SQL and MAC-SQL will invoke the LLM multiple times for each question. They will even utilize the LLM to scan the full database schema. Consequently, they consume large numbers of tokens and have higher delays. As for Gen-SQL, it balances performance and efficiency. On Spider, Gen-SQL improves DIN-SQL by 4.1% in terms of EX, using only 16.5% of tokens and 17.4% of time. On Spider-M, Gen-SQL improves DAIL-SQL by 14.1% in terms of EX, using only 56.5% of tokens, despite an 18.3% relative increase in time. These suggest that our method is *efficient*.



(a) Top-5 (b) Top-10 (c) Gen-SQL (d) Non-iterative Gen-SQL

Figure 4: Distribution of correct samples on BIRD-M. Llama-3-70B is used as the backbone.

Table Retrieval & Text-to-SQL. The relationship between table retrieval performance and downstream Text-to-SQL performance is still understudied. Table 6 shows that EX is not related to F1. Instead, EX is more related to recall. Indeed, our intuition is that the LLM can only generate correct SQL if all relevant tables are retrieved. This intuition can be supported by Figure 4 which presents the distribution of correct samples for each method. The result implies that including a few irrelevant tables is also acceptable for the LLM. So the principle is to maintain higher recall for the retriever, when the retrieved irrelevant tables do not compromise the Text-to-SQL performance.

5 Related Work

5.1 Text-to-SQL

The field of Text-to-SQL has witnessed several paradigm shifts (Kim et al., 2020; Liu et al., 2023). Traditional deep learning based methods utilize Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) for Text-to-SQL (Iyer et al., 2017; Zhong et al., 2017; Dong and Lapata, 2018; Sun et al., 2018; Huang et al., 2018; Bogin et al., 2019; Guo et al., 2019). After the Transformer architecture (Vaswani et al., 2017) has revolutionized the field of natural language processing (Devlin et al., 2019; Raffel et al., 2020), RNN or LSTM modules can be replaced by pretrained Transformer-based models (Wang et al., 2020; Chen et al., 2021b; Scholak et al., 2021; Qi et al., 2022). More recently, the success of LLMs has enabled training-free Text-to-SQL through in-context learning (Brown et al., 2020) where LLMs can learn from a few demonstrations provided in the context to generate SQL (Rajkumar et al., 2022). Due to strong instruction-following ability (Ouyang et al., 2022; Wang et al., 2023) of LLMs, more intricate prompting techniques have been developed to further improve the performance (Wang et al., 2024; Pourreza and Rafiei, 2023; Jiang et al., 2023). However, previous works have ignored the fact that sophisticated prompt or extremely large context containing full database schema will degrade performance. Considering the capability and context limit of LLMs, accurate schema retrieval is a necessity.

5.2 Retrieval-Augmented Generation

LLMs are known to hallucinate facts or fabricate false information (Ji et al., 2023), and have difficulty in timely updating latest or domain knowledge (Jang et al., 2022; Zhou et al., 2023). To remedy these limitations, Retrieval-Augmented Generation (RAG) has been extensively explored (Lewis et al., 2020; Izacard et al., 2022; Shi et al., 2023). RAG is closely related to our task because we both concentrate on accurate retrieval and utilizing the retrieved content. To improve retrieval accuracy, straightforward solutions include re-ranking (Sun et al., 2023) with LLMs, which will typically impact efficiency; and query rewriting (Ma et al., 2023), which requires supervised signals for the retriever. To make use of the retrieved content, Fusion-in-Decoder (FiD; Izacard and Grave, 2021; Zhang et al., 2023) has been proposed to separately encode each retrieved document by an encoder and fuse the encoded documents in the decoder. But this architecture is mainly applied to encoder-decoder models such as BART (Lewis et al., 2019) and T5 (Raffel et al., 2020), and requires additional training. By contrast, the proposed method is applicable to various LLMs without further fine-tuning.

6 Conclusions

This paper proposes Gen-SQL, where the embedding-based retriever is guided by the LLM for efficient and accurate schema retrieval. In the experiments, Gen-SQL exhibits competitive performance on public benchmarks, which shows its effectiveness. On datasets with larger databases, Gen-SQL significantly outperforms the existing

baselines, which proves its scalability. The relationship between table retrieval and Text-to-SQL performance further reveals that the retriever should prioritize recall over precision (to some extent) for better Text-to-SQL performance.

Limitations

Despite the fact that Gen-SQL has achieved competitive results on public benchmarks using Llama-3-70B and GPT-4, its performance on more proprietary LLMs remains to be explored. Since Gen-SQL is a general framework, various LLMs and many other existing Text-to-SQL methods may benefit from Gen-SQL. We leave the investigation of improvements by Gen-SQL on more capable LLMs as our future work.

Acknowledgments

The work is partially supported by the Fundamental Research Funds for the Central Universities 2232023D-19.

References

Christopher Baik, H. V. Jagadish, and Yunyao Li. 2019. Bridging the semantic gap with sql query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 374–385.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565, Florence, Italy. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021a. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu. 2021b. ShadowGNN: Graph projection neural network for text-to-SQL parser. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5567–5577, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,*

Volume 2 (Short Papers), pages 732–738, New Orleans, Louisiana. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Atlas: Few-shot learning with retrieval augmented language models. *Preprint*, arXiv:2208.03299.

Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. 2022. TemporalWiki: A lifelong benchmark for training and evaluating ever-evolving language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6237–6250, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12).

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.

Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to sql: where are we today? *Proc. VLDB Endow.*, 13(10):1737–1750.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *Preprint*, arXiv:2309.06180.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *Preprint*, arXiv:1910.13461.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3).

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023a. Graphix-t5: mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

Jinyang Li, Binyuan Hui, GE QU, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *Preprint*, arXiv:2303.13547.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore. Association for Computational Linguistics.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. *Preprint*, arXiv:1912.01703.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. *Preprint*, arXiv:2402.01117.

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

Bowen Qin, Lihan Wang, Binyuan Hui, Bowen Li, Xiangpeng Wei, Binhua Li, Fei Huang, Luo Si, Min Yang, and Yongbin Li. 2022. SUN: Exploring intrinsic uncertainties in text-to-SQL parsers. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 5298–5308, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

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

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. 2023. Replug: Retrieval-augmented black-box language models. *Preprint*, arXiv:2301.12652.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937, Singapore. Association for Computational Linguistics.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware SQL generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 361–372, Melbourne, Australia. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

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

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang,

Di Yin, Xing Sun, and Zhoujun Li. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql. *Preprint*, arXiv:2312.11242.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

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

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. 2023. Merging generated and retrieved knowledge for open-domain QA. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4710–4728, Singapore. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *Preprint*, arXiv:1709.00103.

Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. 2023. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*.

A Schema Parser

We begin introducing our schema parser module with two illustrative examples.

Consider the following SQL query:

```
SELECT X.A, Y.B, C FROM X, Y
```

Columns `X.A` and `Y.B` are prefixed with their respective table names `X` and `Y`, and they are called *qualified* columns. Column `C` does not explicitly specify its table name, and it is thus *unqualified*. Because there is no way of knowing which table(s) unqualified columns belong to, we append column `C` to both tables and the pseudo-schema is:

- `X (A, C)`
- `Y (B, C)`

A more complex example would be a query containing subqueries:

```
SELECT E FROM Z  -- Scope 3
 WHERE F NOT IN (
   SELECT A FROM X  -- Scope 1
    WHERE B = C
 ) AND G > (
   SELECT max(D) FROM Y  -- Scope 2
 )
```

In this example, all the columns are unqualified. To facilitate a fine-grained table-column assignment, we leverage the concept of *scope* which denotes the context of a `SELECT` statement, and process the scopes one by one. However, for some columns, we still have difficulty in locating the exact tables. For instance, we cannot distinguish whether column `B` or `C` comes from table `X` or `Z`. So the pseudo-schema for the above query is:

- `X (A, B, C)`
- `Y (D)`
- `Z (E, F, G, A, B, C, D)`

Based on these observations, we present the schema parsing algorithm in Algorithm 1. Firstly, the input SQL $\bar{y}$ is parsed into AST (line 1). Secondly, we initialize result set for each table (lines 2-4). Thirdly, we visit and process each scope based on the AST (lines 5-17). Specifically, for

Algorithm 1: Schema parsing algorithm.

Input: (Pseudo-) SQL $\bar{y}$.
Output: Pseudo-schema $\bar{S}$.

```
1  Parse the SQL ȳ into AST;
   /* Initialize the result sets.                    */
2  foreach table ∈ AST.tables do
3  |  R[table] ← ∅;
4  end
5  foreach scope ∈ traverse_scope(AST) do
6  |  Q ← ∅ ; // Unqualified columns in this
   |    scope.
7  |  foreach column ∈ scope.columns do
8  |  |  if column is unqualified then
9  |  |  |  Q ← Q ∪ {column};
10 |  |  else
11 |  |  |  R[column.table] ←
   |  |  |    R[column.table] ∪ {column};
12 |  |  end
13 |  end
   |  /* Append unqualified columns to each
   |     table in this scope.                        */
14 |  foreach table ∈ scope.tables do
15 |  |  R[table] ← R[table] ∪ Q;
16 |  end
17 end
18 Format pseudo-schema S̄ based on R;
19 return S̄;
```

qualified columns, their tables can be directly determined (line 11), and for unqualified columns, they are appended to all the tables in the current scope (lines 14-16). Finally, the results are formatted accordingly and returned (lines 18-19).

B Dataset Construction

It can be noticed that some databases are semantically similar and merging them may cause ambiguity. For example, if we merge two databases both storing information about college students into a single database, there will be two tables about students, resulting in two possible answers to the question asking about the total number of students. To avoid ambiguity, we merge databases based on their similarities. Specifically, for each database, we try to merge it with other databases. Two databases are merged only if the maximum pairwise similarity between their table schemas is less than a predefined threshold so as to prevent merging semantically similar databases. We set the threshold to be 0.75 for both benchmarks.

C Prompts

We retrieve 8 in-context demonstrations based on the cosine similarities between the **{final question}** and the questions from the training set. The stop words are set to {"Question", "Instruction", "\n\n"}.

C.1 SQL Writer

The prompt for SQL generation with database schema:

```
Instruction: Write a sqlite3 SQL
 query to answer the question.

Question: {question 1}
SQL: {SELECT statement 1}

Question: {question 2}
SQL: {SELECT statement 2}

...

Question: {question 8}
SQL: {SELECT statement 8}

Database Schema:
{CREATE TABLE statement 1}
{CREATE TABLE statement 2}
...

Question: {final question}
SQL: SELECT
```

The prompt for SQL generation without database schema:

```
Instruction: Write a sqlite3 SQL
 query to answer the question.

Question: {question 1}
SQL: {SELECT statement 1}

Question: {question 2}
SQL: {SELECT statement 2}

...

Question: {question 8}
SQL: {SELECT statement 8}

Question: {final question}
SQL: SELECT
```

C.2 SQL Debugger

The prompt for fixing SQL queries that fail to execute:

```
Instruction: Write a sqlite3 SQL
 query to answer the question.

Question: {question 1}
SQL: {SELECT statement 1}

Question: {question 2}
SQL: {SELECT statement 2}

...

Question: {question 8}
SQL: {SELECT statement 8}

Database Schema:
{CREATE TABLE statement 1}
{CREATE TABLE statement 2}
...

You may answer the question by
 fixing the SQL that failed to
 execute: {failed SELECT statement}
Error Message: {error message}

Question: {final question}
SQL: SELECT
```