PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency

Zhishuai Li[1,*], Xiang Wang[1,*], Jingjing Zhao[1,*], Sun Yang[2,*], Guoqing Du[1,*], Xiaoru Hu[1,*], Bin Zhang[1,3,4,*], Yuxiao Ye[1,5,*], Ziyue Li[1,6,(✉)], Hangyu Mao[1,(✉)] Rui Zhao[1,],

[1] SenseTime Research, Shanghai, China
[2] Peking University, Beijing, China
[3] Institute of Automation, Chinese Academy of Sciences, Beijing, China
[4] University of Chinese Academy of Sciences, Beijing, China
[5] Beijing Institute of Technology, Beijing, China
[6] University of Cologne, Cologne, Germany
Correspondance: zlibn@connect.ust.hk and hy.mao@pku.edu.cn

Abstract. Recent advancements in Text-to-SQL (Text2SQL) emphasize stimulating the large language models (LLM) on in-context learning, achieving significant results. Nevertheless, they face challenges when dealing with verbose database information and complex user intentions. This paper presents a two-round framework to enhance the performance of current LLM-based natural language to SQL systems. We first introduce a novel prompt representation, called reference-enhanced representation, which includes schema information and randomly sampled cell values from tables to instruct LLMs in generating SQL queries. Then, in the first round, question-SQL pairs are retrieved as few-shot demonstrations, prompting the LLM to generate a preliminary SQL (`PreSQL`). After that, the mentioned entities in `PreSQL` are parsed to conduct schema linking, which can significantly compact the useful information. In the second round, with the linked schema, we simplify the prompt's schema information and instruct the LLM to produce the final SQL (`FinSQL`). Finally, as the post-refinement module, we propose using cross-consistency across different LLMs rather than self-consistency within a particular LLM. Our methods achieve new SOTA results on the Spider benchmark, with an execution accuracy of 87.6%.

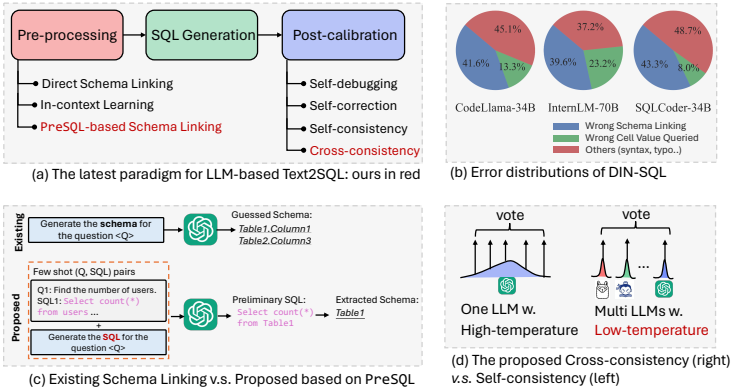
Keywords: Text-to-SQL · Large language models · Prompt · Post-refinement · Cross consistency

1 Introduction

Text-to-SQL (Text2SQL) assists individuals in converting natural language questions (i.e., text) into structure query language (SQL) queries. For example,

* Equal contribution.

conditioned on the contents of the user's question "`How many singers do we have?`" and the corresponding description for the database schema (e.g., table/-column names), the ideal Text2SQL agent can generate SQL statements "`SELECT count(*) FROM singer`", which can then be executed on the SQL database engine to retrieve the desired response. The task is helpful for the database question-answering and information-retrieval applications, such as finance, traffic, and business intelligence [1,2], as it lowers the requirements for expertise and allows non-professional participants to interact with databases in natural language.



(a) The latest paradigm for LLM-based Text2SQL: ours in red

(b) Error distributions of DIN-SQL

(c) Existing Schema Linking v.s. Proposed based on PreSQL

(d) The proposed Cross-consistency (right) *v.s.* Self-consistency (left)

Fig. 1. (a) A three-stage paradigm of SOTA Text2SQL. (b) We summarize the distribution of error types in DIN-SQL [3]. We elaborate on the methods for each type: `PreSQL`-based schema linking for errors in schema linking, cell values references-enhanced prompt for incorrect cell values, and cross-consistency for other errors caused by typos or syntax mistakes. (c) Existing schema linking directly lets LLM guess the schema, we let LLM first generate a preliminary SQL and parse table/column entities from `PreSQL`, increasing accuracy by **+4%**. (d) Our proposed cross-consistency enables the LLMs to operate at low temperatures, thereby reducing hallucination.

The development trajectory of Text2SQL has gone from rule-based, domain-specific solutions to deep learning-based models, specifically sequence-to-sequence models. Recent large language models (LLM)-based solutions have further pushed the performance to a new high. The recent state-of-the-art high-performers [3,4] have decomposed the Text2SQL into a series of subtasks. Such a task decomposition has become a new paradigm, and we generalize it as three stages, as shown in Fig. 1(a): (1) **pre-processing**, including prompt engineering such as zero-shot [5] or few-shot prompting [3], schema linking [3,6], in-context-learning (ICL) [3], (2) **SQL generation**, which is based on LLMs such as CodeLlaMa, ChatGPT, GPT-4, (3) **post-calibration**, including self-consistency [4], self-debugging [1,7], self-correction [3]. Most of the improvements and designs are happening in the preprocessing and post-calibration, particularly in schema linking (+4% in DIN-SQL with CodeX Davinci) and self-consistency (+0.4% in DAIL-SQL with GPT-4).

However, when handling extensive databases across various domains with different formats and intricate user intentions, there is still large room for improvement in preprocessing and post-calibration.

According to the statistics on the Spider [8], an opensource benchmark database for Text2SQL with 10,181 questions and across 200 complex databases in 138 domains with multiple tables, covering almost all important SQL components including GROUP BY, ORDER BY, HAVING and nested queries, around 60.7% questions are about "`WHERE =`" or "`HAVING =`". However, due to various cell values being formatted, an SQL can be semantically correct but yield incorrect results after execution. Take a SQL question of querying gender as an example, male can be formatted variously as '*sex='Male''*, '*sex='male''*, or '*sex='M''*, and there are more examples from our experiences, such as 1000 also being '1k' or '1000.0': Such a formatting variance is even more commonplace and critical in real industrial databases, and misaligned formatting in SQL will produce different results or even fail. Thus, in the prompting, we proposed a **cell value reference**, which provides standardized references in filling out different tables and helps LLM understand the formats and specifications of the database, and in our experiment, it can improve the execution accuracy by maximum +**4%**.

Furthermore, schema-linking, indicating and narrowing down to which tables and which columns the LLMs should search for the answer, is a critical sub-task and performance-booster for Text2SQL. Although works like DIN-SQL [3] proved that directly using LLMs to infer the tables and columns of interest can work better than an end-to-end trained solution, such as BERT [9], we claim that LLMs even GPT-4, have **NOT** been trained or supervised fine-tuned with such a domain-specific schema linking task (i.e., input a specific SQL question and output the relevant table names and column names), as it falls outside the mainstream scope of LLMs. As a result, the schema linking still accounts for about 22% of the mistakes in GPT-4, while the conditions in other LLMs are more serious (see Fig. 1(b)). We innovatively proposed a preliminary-SQL (`PreSQL`)-based schema linking, in which we first generate a rough SQL statement by the LLMs and then parse the mentioned table/column entities as the linking results (see Fig. 1(c)). The rationale behind this is that code (particularly SQL) generation, is a fundamental task across nearly all LLMs, and they may be adequately pre-trained in relevant corpus. Therefore, for LLMs, generating `PreSQL` is easier and more natural than directly performing schema linking as instructed. Consequently, our `PreSQL`-based schema linking achieves maximum +**4%** higher execution accuracy.

Last but not least, two major designs in post-calibration, self-debugging and self-consistency, either have very trivial effects or create more instability in the results, according to our findings. According to our experiences with multiple LLMs, we found that self-debugging is useful for fixing syntax errors, but it tends to be powerless in front of the SQLs generated by strong LLMs, whose problem is semantic ambiguity rather than syntax errors. In terms of self-consistency, the technique that is proposed by the current top-1 opensource solution, is theoretically yielding suboptimal SQL due to its nature: it achieves self-consistency by calling the same LLM several times at a **higher temperature** and choosing the majority as the final output, which firstly does not fundamentally diversify the SQL outputs, and moreover, the higher temperature generates more hallucinations, potentially yielding a stable but bad output. We instead

innovatively propose a "cross-consistency" (in Fig. 1(d)), which calls multiple different LLMs at a low temperature, guaranteeing diversity (due to different LLMs) and further converging to a high-quality result (due to low temperature). We also further proposed two voting schemes, which are (1) native majority voting of the executed results and (2) difficulty-aware voting with distinct candidate LLMs according to the complexity grades of `PreSQL`. Overall, the method achieves state-of-the-art in the Spider [8] leaderboard with 87.6% execution accuracy, being ranked as the top-1 open-source solution. The main contributions are threefold:

- An elaborated prompt is designed and its effectiveness is systemically evaluated under various LLMs. Besides the schema information, the cell values in tables and performance-enhancing instructions are attended to as the prior knowledge in the prompt.
- We explore the `PreSQL`-based schema linking instead of prompting the LLM to generate table/column names directly. We believe it can yield more concise results and is suitable for coding LLM. Coupled with linked schema, we present to organize the simplified prompt and then feed it into the LLM again.
- The cross-consistency is proposed to utilize the diversity across different LLMs. The `PreSQL` is also re-used to vote in this module. Overall, by voting across various LLMs, the final predicted SQL achieves 87.6% execution accuracy in the Spider benchmark.

2 Related Work

Learning-based agents. With the advancement of language models in previous years, multiple methods have been proposed for the Text2SQL task. These early approaches focus on pattern matching between natural language (NL) and SQL statements [10,11]. Some studies employ relation-aware self-attention mechanisms as encoders to learn representations of questions and schemas. These methods then utilize grammar-based decoders to generate SQL as abstract syntax trees or employ sketch-based decoders to obtain SQL by filling in slots [12,13]. Subsequently, the advent of pre-trained language models and the fine-tuning paradigm significantly influences these methods [14]. Numerous studies utilize standard sequence-to-sequence models with transformer architectures [15] to translate NL questions into SQL queries in an end-to-end manner [16,17].

LLM-based in-context learning methods. The development of LLMs has catalyzed substantial transformations in this field. These methods leverage the in-context learning, semantic understanding, and reasoning capabilities of LLMs [18,19], continuously pushing the boundaries of performance on various evaluation benchmarks, such as Spider [8] and BIRD [20]. For example, C3 [5] achieves a breakthrough by leveraging ChatGPT's zero-shot learning capability. DIN-SQL [3] decomposes the task into multiple components, including schema linking, difficulty classification, and SQL generation. This methodology effectively reduces the complexity of decision-making, enabling LLMs to generate more precise SQL statements. DAIL-SQL [4] further enhances the capabilities of LLMs through in-context learning and supervised fine-tuning schemes. In

addition, several other studies incorporate advanced reasoning methods, such as chain-of-thought [21] and self-reflexion [22], to enhance the capabilities of LLMs and improve their performance on Text2SQL tasks.

3 Methodology

For a natural language question Q on a database D, the objective of LLM-based Text2SQL is to translate Q into an executable SQL query s. The likelihood of an LLM $\mathcal{M}$ generating a SQL query s is defined as a conditional probability distribution, where $\mathcal{I}(D)$ is the description of D, $\mathcal{P}$ is the prompt template, and $|s|$ is the length of s. s_i and $s_{<i}$ are the i-th token and the prefix of s_i:

$$\mathbb{P}_{\mathcal{M}}(s|\mathcal{P}(Q,\mathcal{I}(D))) = \prod_{i=1}^{|s|} \mathbb{P}_{\mathcal{M}}(s_i|\mathcal{P}(Q,\mathcal{I}(D)), s_{<i}), \tag{1}$$

As shown in Fig. 2, our PET-SQL framework mainly includes: (1) an elaborated prompt that harnesses the customized instructions, basic database information, and samples in the stored tables (2) instructing the LLM to generate `PreSQL`, in which some demonstrations are selected from pools using a question similarity-based strategy and then prefixed to the prompt as the few-shot in-context (3) finding question-related tables (schema linking) based on the `PreSQL` and prompt LLM to yield FinSQL by the linked schema (4) ensuring consistency in the predicted results across multiple LLMs.

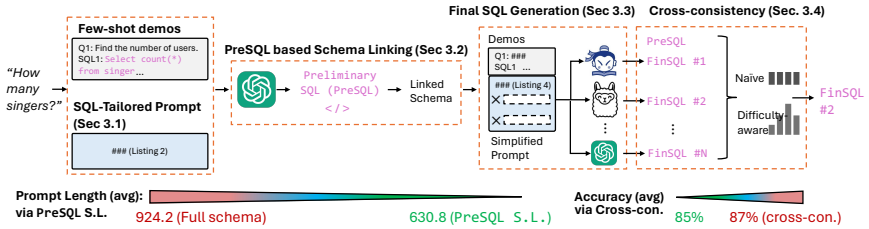


Fig. 2. The overview of PET-SQL framework. It first elaborates on an effective SQL-Tailored prompt (Sec. 3.1), which can be enhanced with selected demos as few-shot in-context learning. Then, `PreSQL` is generated by LLMs for parsing linked schema (Sec. 3.2), where linked entities are retained while unrelated parts are removed from the prompt, thereby simplifying it. After that, the Final SQL can be obtained from LLMs with the concise prompt (Sec. 3.3). Finally, cross-consistency is proposed to post-refine the results generated from multiple LLMs (Sec. 3.4). Our `PreSQL`-based schema linking shortens the prompt length by 32%, and our cross-consistency boosts the Text2SQL accuracy by 2%.

3.1 Pre-processing: SQL-Tailored Prompting

When instructing LLMs to generate SQL queries, the usage of styles or templates for prompts significantly impacts LLMs' performance. As [4] recommended, **C**ode **R**epresentation (CR_p) and **O**penAI

```
OpenAI Demonstration Format:
### How many singers do we have?
SELECT count(*) FROM singers

Code Representation Format:
/* How many singers do we have? */
SELECT count(*) FROM singers
```

Listing 1. OpenAI Demonstration v.s. Code Representation.

Demonstration (OD_p) forms mixed with additional information are better choices (shown in Listing 1). To further exploit the potential of LLM, we enrich the prompt based on OpenAI Demonstration (OD_p), and we name our proposed method as **R**eference-**E**nhanced representation (RE_p).

Building upon OD_p consisting of the task instruction, database schema, and question components, our proposed RE_p performs three modifications: optimization rule (OR), cell value references (CV), and foreign key declarations (FK), which all turn out effective.

Optimization Rule (OR): Concretely, in the task instruction, we raise a multi-task constraint rule for LLM, that is, highlighting the LLM to "minimize SQL execution time while ensuring correctness" (line 1 in Listing 2). The rationale is not only to focus on execution accuracy but also to make LLM aware of the efficiency of SQL statements. During the generation of efficient SQL statements by LLM, redundant characters and operators can be avoided, which often result in exceptions. As our experiment shows, such an optimization rule can boost the performance by as high as +**2.3%**.

Cell Value References (CV): As mentioned before, to avoid mistakes from wrong-formatted cell value, three rows in each table are randomly sampled and inserted in the prompt (lines 7-11 in Listing 2). The sampled cell values as references can help LLM understand the database formats and specifications. It relieves the dilemma of being unsure which element to query as the condition caused by the lack of standardization in filling out different tables. Without prior knowledge of cell values, organizing condition statements to query the number of a particular entity in a table can be confusing. Such a cell value reference can boost performance by +**4.0%**.

```
1  ### Answer the question by SQLite SQL query only and with no
   ↳ explanation. You must minimize SQL execution time while ensuring
   ↳ correctness.
2  ### Sqlite SQL tables, with their properties:
3  #
4  # singer(Singer_ID,Name,Country,Song_Name,Song_release_year,Age,
   ↳ Is_male);
5  # singer_in_concert(concert_ID,Singer_ID);
6  #
7  ### Here is some data information about database references.
8  #
9  # singer(Singer_ID[1,2,3],Name[Joe,Timbaland,Justin Brown],Country[
   ↳ Netherlands,United States,France],Song_Name[You,Dangerous,Hey Oh]
   ↳ Song_release_year[1992,2008,2013],Age[52,32,29],Is_male[F,T,T]);
10 # singer_in_concert(concert_ID[1,1,1],Singer_ID[2,3,6]);
11 #
12 ### Foreign key information of SQLite tables, used for table joins:
13 #
14 # singer_in_concert(Singer_ID) REFERENCES singer(Singer_ID);
15
16 ### Question: How many singers do we have?
17 ### SQL:
```

Optimization Rule

Cell Value Reference

Foreign Key

Listing 2. The proposed SQL-Tailored prompt, containing OR, CV, and FK. It is zero-shot without demos.

Foreign Key Declarations (FK): Thirdly, foreign key relations in the schema are added as suffixes in the prompt (lines 12-15 in Listing 2). It facilitates LLM in recognizing the connections between the tables involved in the database, thus improving the understanding of the user's intent and automatically selecting the appropriate connection in the queries. Such foreign key can boost the performance by as high as +**7.5%**.

The prompt above is in a zero-shot setting and can be further enhanced by the few-shot approach. We will explain how to select demonstrations as shots, equip them with the prompt, and stimulate the LLM to yield the preliminary SQL (`PreSQL`) for a question. Such a `PreSQL` is a critical design for our schema linking with higher accuracy.

3.2 Pre-processing: To Generate A Preliminary SQL (PreSQL) for Schema Linking

To eschew the verbose information in the schema that may obstruct the LLMs' performance, we further conduct the schema linking (SL), which identifies and narrows down the table and column references that are related to the database schema and condition values related to the natural language question. Schema linking has been proven to boost the accuracy, enhance generalizability across domains, and facilitate the synthesis of complex queries [5,3].

However, existing methods directly employed LLMs to "guess" the schema, and such a "guess" can be risky since **LLMs, even GPT-4, have NOT been pre-trained or supervised and fine-tuned with such a domain-specific schema linking task**. Though not being trained with schema linking, most LLMs, luckily, are trained with code generation tasks, such as SQL generation. So our intuitive yet very innovative solution is: **can we solve the schema linking task (what LLM is not good at) by letting LLM generate SQL code (what LLM is good at)?** Thus, rather than instructing the LLMs to link schema, we instruct LLMs to generate a preliminary SQL (`PreSQL`), and then the table and column entities mentioned in the PreSQL are parsed as the linking results. To ensure more accurate schema linking, we incorporate a few steps to generate the `PreSQL`:

Step 1 - Question De-semanticization[23]: the domain-related tokens (i.e., table names '`singer`', column names '`gender`', and values '`male`') in questions are masked with a special token `<mask>` according to the database schema, and thus obtain the question skeletons that only exhibit the question's intentions.

Step 2 - Demo Retrieval based on Similarity: Then, all the question skeletons and question-SQL pairs in the training set are constructed as a demonstration pool. Based on the semantic embeddings of question skeletons, we retrieve the top-K similar samples from the pool with the target question. Like [4], the embedding model for question skeletons is a pre-trained sentence Transformer.

Step 3 - Few-shot demos + SQL-Tailored Prompt: After that, we organize the selected demonstrations with our prompt RE_p following [4], that is, prefixing question-SQL pairs as the few-shot contexts in RE_p (as shown in lines 1-10 in Listing 3). Finally, the few-shot RE_p is used to prompt the LLM and generate `PreSQL`.

Step 4 - Schema Parsing: Then, the SL is implemented based on a simple principle: *The schema information mentioned in the `PreSQL` is concise and relevant to the question*, and thus, the table/column entities in the statement can be directly parsed as the linking results.

Rather than designing strategies such as sorting [5] or extraction [3] to make LLM output relevant database

```
### Some example pairs of questions and corresponding
SQL queries are provided based on similar questions:

### How many farms are there?
SELECT count(*) FROM farm

### How many books are there?
SELECT count(*) FROM book

### How many actors are there?
SELECT count(*) FROM actor
```

Selected Few-shot demos

+

(Listing 2)

Listing 3. Three question-SQL pairs are retrieved and prefixed as a 3-shot prompt, followed by Listing 2 to serve as the full prompt feeding into LLM for `PreSQL`.

references, it is better to generate complete `PreSQL` statements directly. It is also proven in Table 6, via such a `PreSQL` pipeline, our execution accuracy can be boosted by as high as **+12.6%**, compared with the traditional methods that rely on LLM to directly link schema.

There are three advantages: (1) Most LLMs are pre-trained with Text2SQL or SQL-related corpus, so presumably, their ability on SQL generation is stronger than the schema linking output (e.g., a list with `table.column` format) following instructions. (2) Generating SQL is more generic and applicable for coding-specific LLMs such as CodeLlama [24]. (3) The execution accuracy is theoretically monotonic. For a correctly executed `PreSQL`, the linked results are always correct. Therefore, the resulting final SQL (`FinSQL`) should also be correct. Wrong SQL may be attributed to the LLM being confused by the verbose prompt, and there is room to be correct with a concise and simplified prompt. It should be noted that column linking has poor fault tolerance and may be inapplicable to weak LLMs, so we only perform table linking at this stage.

3.3 SQL Generation: To Generate the Final SQL (`FinSQL`)

The Final Prompt: After the schema linking, the mentioned tables and columns in `PreSQL` are utilized to reorganize and simplify the RE_p prompt. Specifically, all the contexts that are irrelevant to the linked tables or columns in the prompt are removed adaptively, including the schema properties, database references, and foreign key declarations: e.g., in Listing 2, if only the table '`singer`' is linked and the table '`singer_in_concert`' not, then all the lines related to the table '`singer_in_concert`' (Line 5, 10-14) can be pruned. With the parsed schemas, the prompt is much simplified. According to our statistics, our tokens in prompt can be saved as much as **31%** after the pruning of `PreSQL`-schema linking.

SQL Generation: To summarize, we first feed full schema information (all the tables and columns) following the Listing 3 into a strong LLM (e.g., GPT-4) to obtain a `PreSQL`, and parse the `PreSQL` for schema linking. Then, the prompt is simplified by pruning irrelevant schemas and used to instruct one LLM or multiple LLMs to generate the `FinSQL`. We will introduce how to orchestra multi-LLMs.

3.4 Post-calibration: Cross-Consistency Is Better than Self-Consistency

As a canonical trick, the self-consistency approach is widely employed for post-refining the executed results, which is conducted by increasing the randomness of an LLM output at high temperatures to generate diverse results and followed by majority voting among multiple executed results. However, it has been reported that high temperatures may hurt the performance of LLMs (increasing model hallucinations) [25], especially for deterministic tasks (e.g., coding). Besides, the diversity of a single model is usually limited. Instead of conducting self-consistency, we propose instructing several LLMs under lower temperatures to generate stable and high-quality SQLs and then casting votes across the SQLs'

executed results. Such a cross-consistency strategy can diversify the SQL queries and maintain LLMs' performance as low temperatures are set up.

In terms of voting, we propose two feasible implementation strategies for cross-consistency: naive voting across several LLMs and difficulty-aware voting according to PreSQL complexity.

Naive voting across several LLMs: Then, the full schema prompt is simplified and used to instruct several LLMs, each of which generates a `FinSQL` with a low temperature. All the `FinSQL`s and the re-used `PreSQL` are executed with the SQL database engine, and the queried results are obtained. Finally, we check the results of each SQL execution and take the majority of the results as the final answer. The `FinSQL` will be selected randomly, whichever yields the final answer.

Difficulty-aware voting according to `PreSQL` complexity: In addition to the naive voting strategy, we further refine the voting rules based on the difficulty of `PreSQL`. As different LLMs specialize in questions with different complexity grades, putting them together in a voting pool can produce biased results. The complexity of questions is classified into four grades (i.e., easy, medium, hard, and extra) by the abstract syntax trees of `PreSQL` according to [26], and each grade of questions is solved by a distinct set of candidate LLMs for voting. With such fine-grained voting, we can maximize the potential of LLMs and significantly mitigate the voting bias.

4 Experiments

4.1 Experimental Setup

Dataset and Metrics: We evaluate our PET-SQL in Spider [8] benchmark, which is a large-scale cross-domain Text2SQL dataset. It contains 8659 instances in training split and 1034 instances in development split over 200 databases, with non-overlapping databases in each set, and 2147 instances are holdout as the test set across another 34 databases. The evaluation of Text2SQL performance of methods is conducted by execution accuracy (EX), following the official test-suit[7] defined in [26]. Additionally, we also conduct experiments on the BIRD-dev [20] dataset to evaluate the effectiveness and generalization of our methods. EX measures the proportion of questions in the evaluation set where the execution results of both the predicted and ground-truth inquiries are identical, relative to the total number of queries.

Evaluated LLMs: Five LLMs are used to validate the superiority of our PET-SQL, and we report the best results of PET-SQL on the test set, which can surpass rank 2 with about 1% improvement on the Spider leaderboard. The LLMs are: CodeLlama-34B [24], SQLCoder-34B [27], InternLM-70B [28], SenseChat-70B [29], and GPT-4 (version on 2023.06.13) [30]. The first two are specific to coding, especially on SQL, while the others are generic LLMs.

[7] `https://github.com/taoyds/test-suite-sql-eval`

Implementation details: To decrease the randomness of LLMs output, the temperatures are set extremely low (10^{-7} for all the non-OpenAI LLMs and 0 for GPT-4). The max lengths of input and output tokens are 4096 and 200, respectively. In the question skeleton-based demonstration retrieval, we traverse all the database schema and check if the table or column names appear in questions. The matched entities will be masked by `<mask>` token. Based on the similarities between the sentence embedding of the target and candidate question skeletons, top-9 similar question-SQL pairs in the demonstration pool are selected and then constructed as the in-context. For cross-consistency, the executed results from the above five LLMs and the `PreSQL` are voting with equal weights.

4.2 Overall Performance

Spider Leaderboard In Table 1, we report the performance of our method and baselines on the Spider test dataset. For naive voting, the executed results from the above five LLMs and the `PreSQL` are voting with equal weights, while the selection strategy of LLMs in difficulty-aware voting is illustrated in Table 10. Since the leaderboard is closed and rejects new submissions, we tested the performance of our approach offline, with the official test-suit. Thus, it can be concluded that our method achieves the highest execution accuracy among all non-learning-based methods. Specifically, our method surpasses DAIL-SQL by 1% and achieves the best among the open-source methods in the leaderboard [8]. Even with the naive voting strategy, the superiority of our PET-SQL still holds.

Table 1. The performance on the Spider leaderboard

Methods	**EX**
RESDSQL-3B + NatSQL [31]	79.9%
C3 + ChatGPT + Zero-Shot [5]	82.3%
MAC-SQL + GPT-4 [7]	82.8%
DIN-SQL + GPT-4 [3]	85.3%
DAIL-SQL + GPT-4 + Self-consistency [4]	86.6%
PET-SQL (Naive voting)	86.7%
PET-SQL (Difficulty-aware voting)	**87.6%**

Comparison under other foundation LLMs We further validate the performance difference and consistency between our PET-SQL and other top 2 methods (DAIL-SQL, DIN-DQL) when using each different foundation LLM.

Our general framework and SQL-tailored prompt are universally effective for various LLMs, even without schema linking and cross-consistency: In this setting, only one LLM (specified in each column) attends to the Text2SQL task during the comparison. Thus, we remove our schema linking and cross-consistency modules in the framework since their implementations involve GPT-4 and other LLMs, which may cause unfairness. The results are shown in Table 2. Generally, even without schema linking and cross-consistency, the performance of our approach and DAIL-SQL are mutually exclusive on the LLMs on the dev set. While on the test set, our approach scores an overwhelming

[8] Since the 1st-rank MiniSeek is not publicly available, we do not engage it in comparisons.

Table 2. Comparing top 2 methods under a single LLM (9-shot), thus our scheme linking (SL) and cross-consistency (CC) are removed.

Datasets	Methods	CodeLlama	SQLCoder	InternLM
Spider-dev	DIN-SQL [3]	0.673	0.678	0.686
	DAIL-SQL [4]	**0.756**	**0.709**	<u>0.742</u>
	PET-SQL (w/o SL, CC)	<u>0.750</u>	**0.709**	**0.750**
Spider-test	DIN-SQL [3]	0.638	0.626	0.672
	DAIL-SQL [4]	<u>0.720</u>	<u>0.697</u>	<u>0.707</u>
	PET-SQL (w/o SL, CC)	**0.744**	**0.741**	**0.714**

dominance, achieving 1%∼6% improvements against the DAIL-SQL. All of these verify the superiority of PET-SQL's general framework and prompt design.

4.3 A Deep Dive into Prompt Design

Our few-shot RE_p prompt is better than code representation and OpenAI demonstration: We compare our few-shot RE_p with CR_p and OD_p, which are recommended by [4]. It should be noted that the selected question-SQL demonstrations in them are the same. They are evaluated on the Spider development and test datasets separately, with the same selected question-SQL demonstrations in them. As shown in Table 3, we recap the conclusions here: our proposed prompt RE_p demonstrates superior performance under the zero-shot setting across three different datasets and three different foundation LLMs, with roughly 1% to 7% improvements compared to CR_p. This indicates the versatility of RE_p, whether for coding-specific or generic LLMs.

Table 3. The performance comparison between different prompt styles (0-shot)

Datasets	Prompting	Codellama	SQLCoder	InternLM
BIRD-dev	CR_p	0.333	0.266	0.265
	RE_p	**0.355**	**0.345**	**0.308**
Spider-dev	CR_p	0.685	0.562	0.701
	RE_p	**0.738**	**0.657**	**0.707**
Spider-test	CR_p	<u>0.694</u>	<u>0.631</u>	0.649
	OD_p	0.609	0.579	<u>0.673</u>
	RE_p	**0.717**	**0.678**	**0.699**

Ablation of RE_p: We further scrutinize the effects of three modifications in RE_p compared to OD_p: optimization rule (OR), cell values reference (CV), and foreign key declarations (FK) by ablation studies under the zero-shot setting. As shown in Table 4, (1) **Our cell values reference is the most important**: Discarding it in InternLM and SQLCoder leads to **-2.4%** and **-6.4%** decreases in EX. This indicates that ensuring the presence of sampled cell values is crucial for LLM generation, and with these references, LLM can better navigate the database and

Table 4. The ablation of the RE_p on Spider-dev set (0-shot): the full RE_p highlighted in grey, the most important component in boldface (lowest EX), the second underlined.

	CodeLlama	SQLCoder	InternLM
RE_p	73.80%	65.70%	70.70%
w/o OR	71.50%	64.60%	<u>69.90%</u>
w/o CV	<u>71.30%</u>	**61.70%**	**69.10%**
w/o FK	**66.30%**	<u>62.30%</u>	71.60%

formulate accurate queries. This reduces ambiguity and improves the efficiency of the generated SQL queries. (2) **Optimization rule also always helps for all three models**: Dropping OR also shows a consistent impact on all three models' performances (**-3%**). (3) **Foreign key instead has two-sided impact**: In CodeLlama, it boosts the performance to the most extent by +**7.5%**, but in InternLM, not using FK is surprisingly better.

To conclude, **the CV has the greatest significance, followed by the OR and FK components.** To provide a more reasonable explanation of how the OR component works, we conduct studies on the execution efficiency of generated SQL queries before and after applying the OR. The evaluation metric VES [9] used for assessing the efficiency can be referred to in the BIRD benchmark. The results are reported in Table 5 (higher is better). It can be seen that the use of OR is helpful for the execution efficiency.

Table 5. The VES of generated SQL queries w/ and w/o OR

	CodeLlama	SQLCoder	InternLM
w/o OR	18.94	24.28	22.63
w/ OR	**20.75**	**27.77**	**25.23**

4.4 A Deep Dive into Schema Linking

The linked schema is parsed from the `PreSQL`, generated by GPT-4 with full schema information. To validate its effect, we introduce the table recall metric R_e (higher is better) and compare the EX of `FinSQL`s generated by the LLMs based on simplified and unsimplified schemas separately. $R_e = \sum_{i=1}^{N} \mathbf{1}_e / N$, where N is the number of instances in the test set. $\mathbf{1}_e$ is an indicator function, which returns 1 if the linked tables are exactly the tables that appeared in ground-truth SQL (GT tables) and 0 otherwise. $R_e = 1$ is the ideal schema linking result, which means all the question-related tables are recalled, and there is no verbose information.

Table 6. Comparing schema linking (SL) performance (table recall metric) on Spider dataset between extraction-based [3]/sorting-based [5] SLs and the `PreSQL`-based SL.

Datasets	**SL Methods**	**CodeLlama**	**SQLCoder**	**InternLM**
Spider-dev	Extraction-based SL [3]	0.804	0.815	0.676
	Sorting-based SL [5]	0.855	n/a	0.688
	`PreSQL`-based SL	**0.864**	**0.856**	**0.801**
Spider-test	Extraction-based SL[3]	0.802	0.790	0.701
	`PreSQL`-based SL	**0.827**	**0.840**	**0.827**

To evaluate the superiority of the proposed `PreSQL`-based SL, in Table 6, we compare it with the extraction-based / sorting-based SL, which is used in DIN-SQL[3] and C3[5], respectively. It becomes evident that (1) **the proposed SL can significantly extract useful information**. (2) **It also significantly shortens the prompts.** By using the linked schema, we can cut the average token length of the prompt from 924.2 to 630.8, by more than **33%**.

[9] https://bird-bench.github.io/

In Table 7, we further summarize the average tables mentioned in the prompt and the ground truth (i.e., the average of GT tables). In the full schema (without SL), the average number of tables is 4.89, while in the simplified schema (with SL), it is 1.60 (close to GT). This means that SL significantly simplifies useless table information in prompts.

Table 7. The average of tables in prompts

w/o SL	w/ SL	Ground truth
4.89	1.60	1.57

Table 8. The performance with and without SL (9-shot)

	Codellama	**SQLCoder**	**InternLM**	**GPT-4**
w/o SL	74.4%	74.1%	71.4%	85.2%
w/ SL	**78.2%**	**76.9%**	**75.4%**	**85.5%**

With the linked tables, we further simplify the prompt and then evaluate the performance of LLMs on the test set with the 9-shot setting. The results are demonstrated in Table 8. (1) **The simplified prompt has improved performance across all LLMs** achieving 1%∼6% improvements, especially for CodeLlama, which validates the effectiveness of our `PreSQL`-based approach. (2) **Smaller LLMs tend to benefit significantly from SL while larger LLMs gain slightly.** This is likely due to the inherent differences in their capacities and parameters for retrieving useful information from raw text. In conclusion, SL is an effective technique that enhances the performance and versatility of LLMs by summarizing structured information.

4.5 A Deep Dive into Cross-consistency

In Table 9, we report the performance of cross-consistency (CC) on dev and test sets. Considering the token cost, we give up the SQL generation by GPT-4 on the dev set, and the best result from the remaining LLMs is chosen as the baseline (i.e., CodeLlama). For comparison, we also perform self-consistency with CodeLlama (i.e., calling it 5 times, *temperature* = 0.5).

Table 9. The performance of CC (9-shot)

Methods	**Spider-dev** (w/o GPT-4)	**Spider-test** (w/ GPT-4)
Baseline	75.3% (by CodeLlama)	85.5%
+ Self-consistency	76.1% (by CodeLlama)	-
+ Naive voting	**82.2%**	86.7%
+ Difficulty-aware voting	-	**87.6%**

It can be observed that, (1) **Cross-consistency is much better than self-consistency due to diversity**. On the dev set, the gain of self-consistency is trivial (only 0.8%), while the cross-consistency with naive voting achieves 6.9% improvements. On the test set, the naive voting of CC can already improve +1.2%, while In DAIL-SQL [4], self-consistency (five SQLs generated by GPT-4) only achieves +0.4%. Thus, bringing diverse LLMs can better boost the accuracy since different LLMs may make different errors, and letting them vote mitigates each other's mistakes. (2) **Difficulty-aware voting is a better implementation of**

Table 10. Suggested set of LLMs for difficulty-aware voting w.r.t various difficulty questions on Spider-test

Difficulty	Models for CC	EX
Easy	SQLCoder-34B, InternLM-70B, SenseChat-70B	0.932
Medium	GPT-4, InternLM-34B, SenseChat, CodeLlama-34B	0.907
Hard	GPT-4, InternLM-70B, SenseChat-70B	0.849
Extra	GPT-4, SenseChat-70B	0.759
Total		0.876

CC. Compared to naive voting, the difficulty-aware voting strategy can further exploit the potential of LLMs and significantly mitigate the voting bias, improving the performance from 86.7% to 87.6% on the test set.

For questions in different difficulty levels, a tailored set of multi-LLMs can further optimize the accuracy. Table 10 gives the details of our recommended set of LLMs for each difficulty level and their EX on the test set using the `PreSQL` only. (1) **Smaller LLMs are sufficient to handle the easy questions**: For easy questions, smaller LLMs can achieve 93.2% EX, demonstrating their efficacy in handling less complex questions. (2) **For difficult questions, there is a clear need for more powerful LLM** (i.e., GPT-4) to maintain high performance. This performance disparity underscores the importance of selecting the appropriate LLMs based on the difficulty of the questions.

CC as a plug-and-play can always boost performance: CC can serve as a plug-and-play module for other SOTA methods. We evaluate its performance with DAIL-SQL on the Spider-test set as shown in Table 11. With the introduction of CC, the DAIL-SQL can be further enhanced by **+2%** improvements. This suggests that the CC strategy is still effective on other methods. CodeLlama is used as the single LLM setting, and then SQLCoder and InternLM are added as the CC setting.

Table 11. CC as plug-and-play

	EX
DAIL-SQL	0.720
DAIL-SQL + CC	0.735

4.6 Emphasize the generalizability at other benchmarks

To validate the transferablity and generalizability of the proposed PET-SQL, We compare it with more baselines across other benchmarks/datasets, and the EX is reported in Table 12 (higher is better). It can be seen that our proposed method still outperforms the Spider and BIRD benchmarks.

Table 12. The EX results on Sider-dev and BIRD-dev sets (higher is better)

	Spider-dev	**BIRD-dev**
STF CodeS-15B	84.9	47.9
ACT-SQL [32]	82.9	n/a
DIN-SQL [3]	82.8	50.7
DAIL-SQL [4]	84.4	54.8
DTS-SQL [33]	<u>85.5</u>	<u>55.8</u>
MAC-SQL [7]	**86.8**	**57.6**
PET-SQL (**ours**)	89.3	60.1

5 Conclusion

This paper presents a two-round framework based on pre-trained LLMs, PET-SQL, which aims to address challenges in Text2SQL tasks by enhancing the prompt and leveraging cross-consistency across LLMs. Our approach achieves

87.6% execution accuracy on the Spider leaderboard. We also propose a `PreSQL`-based schema linking method to simplify prompt information and improve the efficiency and accuracy of LLMs in generating SQL queries. Overall, the PET-SQL framework demonstrates promising results and opens avenues for further advancements in Text2SQL tasks.

References

1. Sui, G., Li, Z., Li, Z., Yang, S., Ruan, J., Mao, H., Zhao, R.: Reboost large language model-based text-to-sql, text-to-python, and text-to-function–with real applications in traffic domain. arXiv preprint arXiv:2310.18752 (2023)
2. Zhang, B., Ye, Y., Du, G., Hu, X., Li, Z., Yang, S., Liu, C.H., Zhao, R., Li, Z., Mao, H.: Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. arXiv preprint arXiv:2403.02951 (2024)
3. Pourreza, M., Rafiei, D.: Din-sql: Decomposed in-context learning of text-to-sql with self-correction. Advances in Neural Information Processing Systems **36** (2024)
4. 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. arXiv preprint arXiv:2308.15363 (2023)
5. Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Lin, J., Lou, D., et al.: C3: Zero-shot text-to-sql with chatgpt. arXiv preprint arXiv:2307.07306 (2023)
6. Yang, S., Su, Q., Li, Z., Li, Z., Mao, H., Liu, C., Zhao, R.: SQL-to-Schema enhances schema linking in text-to-sql. In: Database and Expert Systems Applications - 35th International Conference,DEXA 2024
7. Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Zhang, Q.W., Yan, Z., Li, Z.: Mac-SQL: Multi-agent collaboration for text-to-sql. arXiv preprint arXiv:2312.11242 (2023)
8. Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint arXiv:1809.08887 (2018)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
10. Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103 (2017)
11. Xu, X., Liu, C., Song, D.: Sqlnet: Generating structured queries from natural language without reinforcement learning. arXiv preprint arXiv:1711.04436 (2017)
12. Yin, P., Neubig, G.: A syntactic neural model for general-purpose code generation. arXiv preprint arXiv:1704.01696 (2017)
13. Dong, L., Lapata, M.: Coarse-to-fine decoding for neural semantic parsing. arXiv preprint arXiv:1805.04793 (2018)
14. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research **21**(1), 5485–5551 (2020)
15. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)

16. Scholak, T., Schucher, N., Bahdanau, D.: Picard: Parsing incrementally for constrained auto-regressive decoding from language models. arXiv preprint arXiv:2109.05093 (2021)
17. Xu, K., Wang, Y., Wang, Y., Wen, Z., Dong, Y.: Sead: End-to-end text-to-sql generation with schema-aware denoising. arXiv preprint arXiv:2105.07911 (2021)
18. Ruan, J., Chen, Y., Zhang, B., Xu, Z., Bao, T., Du, G., Shi, S., Mao, H., Zeng, X., Zhao, R.: Tptu: Task planning and tool usage of large language model-based ai agents. arXiv preprint arXiv:2308.03427 (2023)
19. Zhang, B., Mao, H., Ruan, J., Wen, Y., Li, Y., Zhang, S., Xu, Z., Li, D., Li, Z., Zhao, R., et al.: Controlling large language model-based agents for large-scale decision-making: An actor-critic approach. arXiv preprint arXiv:2311.13884 (2023)
20. Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., et al.: Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. Advances in Neural Information Processing Systems **36** (2024)
21. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems **35**, 24824–24837 (2022)
22. Shinn, N., Labash, B., Gopinath, A.: Reflexion: an autonomous agent with dynamic memory and self-reflection. arXiv e-prints pp. arXiv–2303 (2023)
23. Guo, C., Tian, Z., Tang, J., Wang, P., Wen, Z., Yang, K., Wang, T.: A case-based reasoning framework for adaptive prompting in cross-domain text-to-sql. arXiv preprint arXiv:2304.13301 (2023)
24. Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al.: Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950 (2023)
25. Renze, M., Guven, E.: The effect of sampling temperature on problem solving in large language models. arXiv preprint arXiv:2402.05201 (2024)
26. Zhong, R., Yu, T., Klein, D.: Semantic evaluation for text-to-SQL with distilled test suites. In: Webber, B., Cohn, T., He, Y., Liu, Y. (eds.) Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 396–411. Association for Computational Linguistics, Online (Nov 2020). https://doi.org/10.18653/v1/2020.emnlp-main.29, https://aclanthology.org/2020.emnlp-main.29
27. SQLCoder, D.: sqlcoder-34b-alpha. https://huggingface.co/defog/sqlcoder-34b-alpha (2023)
28. Team, I.: InternLM: A multilingual language model with progressively enhanced capabilities. https://github.com/InternLM/InternLM (2023)
29. SenseTime: SenseChat. https://platform.sensenova.cn/#/doc?path=/chat/ChatCompletions/ChatCompletions.md (2024)
30. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
31. Li, H., Zhang, J., Li, C., Chen, H.: ResdSQL: Decoupling schema linking and skeleton parsing for text-to-sql. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 37, pp. 13067–13075 (2023)
32. Zhang, H., Cao, R., Chen, L., Xu, H., Yu, K.: ACT-SQL: In-context learning for text-to-sql with automatically-generated chain-of-thought. arXiv preprint arXiv:2310.17342 (2023)
33. Pourreza, M., Rafiei, D.: DTS-SQL: Decomposed text-to-sql with small large language models. arXiv preprint arXiv:2402.01117 (2024)