DFM-SQL: A Multi-Approach Framework with Candidate Selection and Correcting for Text-to-SQL

Anonymous ACL submission

Abstract

To address the challenges of improving the performance of large language models in Text-to-SQL tasks, we propose DFM-SQL, a framework that integrates multiple innovative strategies to significantly enhance the generation and selection of candidate SQL statements. Specifically, we developed a multiple LLMs generator system to produce a diverse and high-quality set of candidate SQL queries. The generator employs two core methods: firstly, a Divide-and-conquer strategy that breaks down complex queries into manageable sub-queries within a single LLM call, and secondly the construction of an In-domain Knowledge Base for the database schema using LLMs to enhance contextual understanding. To ensure the quality of the generated SQL statements, we also developed a dedicated selector agent to refine and select high-quality SQL queries produced by the generator. Additionally, we employed a few-shot learning approach, leveraging LLMs to fine-tune and refine the candidate SQL queries for improved accuracy and performance. Experimental results demonstrate that the DFM-SQL framework not only significantly enhances the quality and diversity of SQL queries, but also substantially narrows the gap between execution accuracy and exact match accuracy. In benchmark tests on the Spider Text-to-SQL dataset, DFM-SQL achieved groundbreaking results: an execution accuracy of 85.3% and an exact match accuracy of 86.3%, with only a 1% difference between the two metrics. This achievement marks a new milestone in the consistency between execution accuracy and exact match accuracy, while also pushing the exact match accuracy to a new SOTA level.

1 Introduction

Text-to-SQL is a natural language processing task that turns natural language into SQL queries. NLP research has been transformed by the fast growth of Large Language Models(LLMs)(Yao et al., 2023). LLMs act as versatile tools for solving language tasks and excel in many NLP applications, including math(Zheng et al., 2024), reasoning(Kojima et al., 2022), and coding(Jiang et al., 2024). However, existing research(Pourreza and Rafiei, 2023b; Liu et al., 2023) shows that LLMs using zero-shot or few-shot(Park et al., 2024) prompts still struggle to surpass carefully optimized specialized models in Text-to-SQL tasks. This is because the task requires meeting multiple complex demands at once, such as semantic alignment, schema understanding, and code generation. Studies have shown that task decomposition is an effective strategy for solving complex tasks with LLMs. This involves breaking down a complex task into simpler subtasks and guiding the LLMs to solve them step by step(Kojima et al., 2022).

Method	**EX(%)**	**EM(%)**
Single Query	70.1	64.6
framework Self-consistency	86.6	70.7
Upper-bound	**84.3**	**85.3**

Table 1: An integrated approach for evaluating single query generation on the Spider test set with achievable self-consistency and upper bounds, where EX stands for Execution Accuracy and EM stands for Exact Match Rate.

Recently, DIN-SQL(Pourreza and Rafiei, 2023a) was proposed for Text-to-SQL, which decomposes the Text-to-SQL task into four subtasks: schema linking, categorization, SQL generation, and self-correction. Then it solves these subtasks using a Chain-of-Thought(COT) prompt. Although task decomposition strategies show promise for complex tasks, current methods like DIN-SQL still face major limitations. For example, their schema linking modules often fail to accurately match problem keywords with relevant data fields, and their self-learning mechanisms are inefficient at correcting errors. Approaches like LPE-SQL's(Chu et al., 2024) self-consistency also suffer from performance gaps as high as 14%. The notable gap between execution accuracy (EX 86%) and exact match rate (EM 70.7%) in DAIL-SQL(Gao et al., 2024), the current best method on the Spider benchmark(Yu et al., 2018), suggests that the candidate query ranking mechanism still has significant room for improvement. To address the above challenges, this paper proposes the DFM-SQL framework, which achieves performance breakthroughs through innovative candidate generation and preference mechanisms. As shown in the upper bound in Table 1, the accuracy of our EM is as high as 85%, and the execution accuracy of EX reaches 84.3%.

Our goal is to create a diverse set of high-quality candidate responses and select the best one through an effective ranking mechanism. Specifically, we propose two different candidate generation methods, each capable of producing high-quality responses. (1)The first approach tackles the schema linking problem by building an In-domain Knowledge Base. We use leading LLMs or manual methods to extract database entity relationships, then validate them manually. This creates a knowledge base with table structures, foreign key constraints, and field semantic annotations. This knowledge base reduces the need for LLMs to learn the database schema and helps manage complex fields and foreign key relationships more effectively. (2)Aiming at the logical nesting problem of complex SQL queries, we propose a COT partitioning strategy, which is first applied to the Text-to-SQL task. The method uses dependency parsing to identify conditional relationships, breaks down nested conditions into simple predicates, generates SQL queries step by step, and then combines them into a complete query.

High-quality and diverse candidate responses are essential for the scoring method, as low diversity reduces comparability and weakens the selection mechanism's ability to assess candidate quality. To address this, we introduce a selection agent that builds a comparison matrix for candidate query and selects the final response with the highest cumulative score, leveraging the strengths of each strategy to significantly boost overall performance. Despite the near-perfect consideration of every detail in our steps, syntax, field, or logic errors may still occur when generating SQL queries. To address this, we introduce a small set of manually crafted correct and incorrect SQL query examples to guide advanced LLMs in making fewer errors, which is crucial for narrowing the gap between EX and EM metrics. The correction program generates queries through reflection, uses error feedback to guide corrections, and applies this iterative process at every critical step. We thoroughly evaluated the DFM-SQL method in the Spider benchmark test. The results show that DFM-SQL increases exact match accuracy from 74% to 85.6% and achieves 86.0% execution accuracy, significantly narrowing the gap with top-performing methods.

In summary, our three contributions are as follows:

- To address the challenge of understanding complex database structures, we propose an In-domain Knowledge Base that makes database information easier for LLMs to learn and manage. To optimize the Divide-and-Conquer approach, we use more detailed strategies for complex SQL queries, such as nested, inferential, mathematical, and multi-table linking, to address various complexity challenges.

- Our selection process takes advantage of the contextual learning abilities of advanced LLMs, trained with different classification goals, to handle the randomness of candidate queries while minimizing SQL queries quality degradation. Errors like syntax, logic, and linking issues are further corrected through few-shot LLMs techniques.

- Experiments show that our system performs well on the Spider dataset, with a precise matching correct rate of 85.6%, exceeding the current state-of-the-art system by 4.3 percentage points. Meanwhile, the precise execution accuracy rate reaches 85.3%, which significantly narrows the gap with the matching rate and improves the consistency between theoretical and practical operations. This enhancement reduces the cost and risk of incorrect queries, and improves query accuracy and efficiency.

2 Related Work

The natural language problem of generating accurate SQL queries, the initial progress involved customizing templates(Zelle and Mooney, 1996), which required a lot of manual work. Earlier approaches utilized converter-based sequence-to-sequence models(Sutskever et al., 2014), well suited for tasks involving sequence generation, including Text-to-SQL(Qin et al., 2022a), but the models are still overstretched for generative tasks. Initial sequence-to-sequence models, such as IR-Net(Guo et al., 2019), use a bidirectional LSTM architecture with self-attention to encode queries and database schemas. For better integration of schema information, models such as RAT-SQL(Wang et al., 2020) and RASAT(Qi et al., 2022) incorporate relation-aware self-attention, while SADGA(Cai et al., 2021) and LGESQL(Cao et al., 2021) use graph neural networks for schema querying relations. Despite these advances, sequence-to-sequence models still lack human-level understanding and do not achieve more than 60% accurate matches on the Spider retention test set.

Along with the growing use of LLMs in various NLP fields, the Text-to-SQL domain has also benefited from recent methodological innovations that use LLMs to enhance performance. Some scholars' approaches(Tan et al., 2024) utilize the zero-sample context learning capability of LLMs to generate SQL. Building on this foundation, subsequent models, including DIN-SQL, DAIL-SQL, MAC-SQL(Wang et al., 2024) and C3(Dong et al., 2023), and other subsequent models improve LLMs performance through task decomposition. In addition to contextual learning, proposals in DAIL-SQL, DTS-SQL(Pourreza and Rafiei, 2024), and CodeS(Li et al., 2024) attempt to improve the capabilities of open-source LLMs through supervised fine-tuning. However, the biggest performance improvements were seen in proprietary LLMs that use contextual learning methods(Li et al., 2023). Unlike previous approaches, this paper introduces an efficient hybrid method that accurately generates superior candidate SQL queries and proposes small-sample correction techniques to leverage the valuable, often overlooked, correct and error information during SQL queries generation.

In addition, our method bridges the gap between the accurate execution rate and the accurate matching rate that was too large in previous methods. In contrast to most previous work, the Distillery approach(Maamari et al., 2024) demonstrates that the latest LLMs can efficiently handle up to 200 columns of database schema information within a hint, eliminating the need for a separate schema-linking step that could introduce errors(Talaei et al., 2024). In this study, we confirm that for benchmarks like Spider, where patterns typically have fewer than 200 columns, pattern linking is unnecessary. Independent of, but concurrent with, our work, CHASESQL(Pourreza et al., 2025) introduces methods that generate a large number of candidate responses for a given problem during inference. We modify the response methods for these candidates so that we only focus on SQL statements that cannot be correctly executed, rather than modifying fully executable SQL statements during training.

3 Methdology

This section outlines the DFM-SQL framework, as shown in Figure 1. (1)Design In-domain knowledge base: Parsing database schemas and building In-domain Knowledge Base. (2)Divide-and-Conquer module breaks down complex queries into subtasks. (3)Candidate Agent Selects Candidate SQL queries via Comparison Matrix. (4)The Few-shot(Park et al., 2024) Correction module iteratively correct syntax errors. During the candidate generation phase, the correction phase ensures that all candidates passed to the selection agent are syntactically valid queries. Additionally, in the final output phase step, it applies semantic corrections to the selected results to resolve deeper issues like field mapping errors.

3.1 In-domain Knowledge Base

Promising results have been achieved using the M presentation for contextual learning with fewer samples across a variety of related tasks(Pourreza and Rafiei, 2023a). A large number of schemas and fields in a database can be hard to understand, but some are crucial, as they are used in various SQL statements, such as retrieving Name and Population from the City table. Building presentations with relevant tables and columns can help the model not only understand underlying data patterns but also specify tasks and illustrate the step-by-step process of deriving outputs. Figure 1 outlines a construction method for generating table interpretations online, starting with generating initial schema interpretations using Qwen2.5-Instruct, and then manually verifying the templates to ensure that an in-domain repository is formed based on the table tables, its Prompt template is shown in Appendix A. These steps allow schemas in SQL queries to be extracted more accurately.

3.2 Divide-and-Conquer

This section explains the Divide-and-Conquer module, which breaks down a complex problem

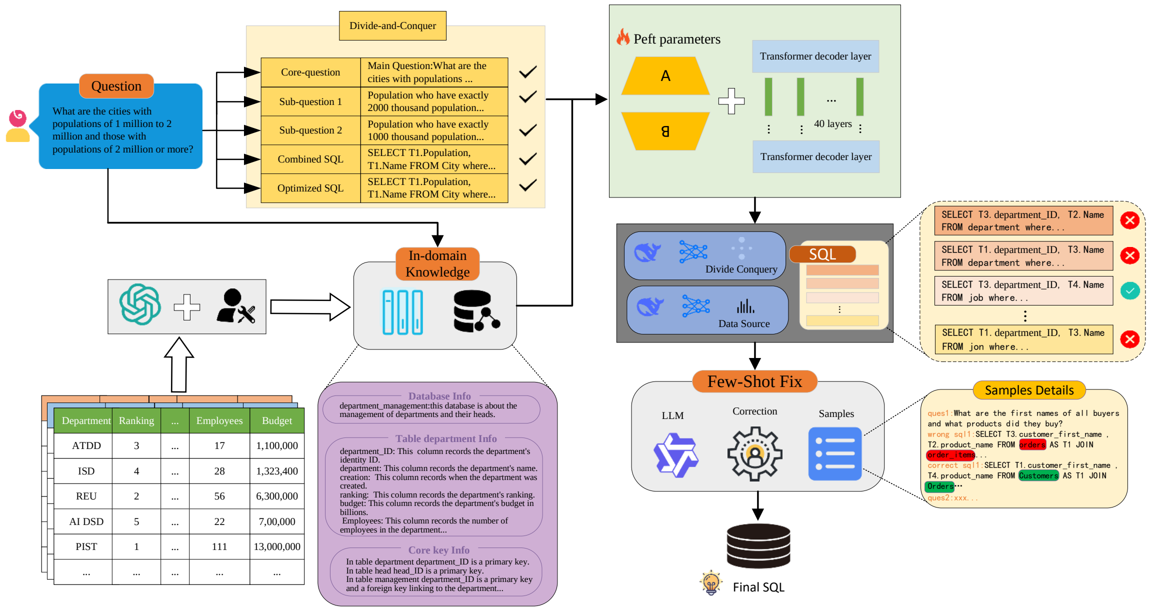

Figure 1: Overview of the DFM-SQL framework for Suggested Text-to-SQL, which uses a candidate agent to pick better answers from the candidate answers generated, while using a correction tools to provide feedback to improve the output.

into smaller sub-problems, solves each separately, and then merges the solutions for the final answer. Along these lines, we propose a CoT hinting approach that first decomposes the given problem into smaller subproblems using pseudo-SQL query examples. The solutions to these sub-problems are then aggregated to construct the final answer. Finally, the constructed query is optimized to remove redundant terms and conditions. We have found this approach particularly effective in handling complex situations, such as nested queries, including intricate WHERE or HAVING clauses, and queries that involve advanced mathematical operations. As in Appendix's Figure 3, we provide an example of a problem and its corresponding SQL query successfully solved using this generator. However, due to the complexity of the conditions and SQL statements of this query, we first solved a problem and a complex SQL query and designed it step-by-step in a hint template as an example.

3.3 Candidate Agent

By using the two methods above to generate SQL queries, we can produce multiple sets of candidate queries for any given problem. The key challenge in this step is selecting the correct SQL query from the candidate pool. A simple approach is to measure the consistency between candidates by executing them, grouping them according to their execution results, and selecting the query from the largest group as the most likely correct answer. The problem with this approach is that it assumes the most consistent answer is the correct one, which isn't always true, as LLMs may learn the wrong features and mistakenly classify a group of incorrect SQL queries as correct, making this majority-based or weighted voting method prone to misclassification. We propose a finer-grained selection strategy, which relies on a selection agent. Given a set of candidate SQL queries $C = \{c1, c2, \ldots . . cn\}$, the final response is selected by identifying the candidate with the highest score, as determined by a selection model. The model θp can take k candidates and rank them according to the accuracy of each of them in answering a given question. We learned from (Pourreza et al., 2025) about selection agent equation, and we changed it when $\mathrm{E_{c1}} = \mathrm{E_{c2}} = \ldots = \mathrm{E_{cn}} = 0$. Specifically, we formalize the selection of the final response as:

$$C = \underset{c \in C}{\operatorname{argmax}} \left(\sum_{i=1}^{(n,k)} \theta_p \left(c_{i1}, c_{i2}, \ldots, c_{ik} \right) \mid Q_\mu, H_\mu, D \right) \qquad (1)$$

Where Q_μ refers to the user's question, H_μ is the prompt provided, and D is the target database where the question was asked. We pass k candidates to the selection model to be ranked, with k between 1 and n. The model is not able to compare the candidates. In the extreme case when $k = 1$, the model is unable to make comparisons between candidates, which complicates the evaluation process of the model. As k increases, comparing more candidates makes the modeling process more challenging. However, having diverse results helps in identifying the exact answer. For example, if one candidate in the test benchmark successfully passes ($\mathrm{E_{ci}} = 1, \mathrm{E_{c1}} = \mathrm{E_{c2}} = \ldots = \mathrm{E_{ci-1}} = \mathrm{E_{ci+1}} \mathrm{E_{cn}} = 0$), it is the only correct answer, eliminating the need to compare it with incorrect ones, though this is true for the vast majority of cases. The most straightforward solution to ensuring a high-quality and diverse set of candidates is using off-the-shelf LLMs for pairwise selection, but since candidates are often very similar, a fine-tuned model is needed to capture nuances and make more accurate predictions. To train the selection agent, we first train and generate candidate SQL queries scaled on the training set (Text-to-SQL benchmark) and categorize them into clusters based on their execution results. We also consider that to avoid order bias during training, we randomize the order of correct and incorrect queries in each pair. Since the number of cases with both correct and incorrect candidates is limited, for cases where no correct candidate exists, we include a basic real SQL query as a hint to guide the model in generating the correct candidate.

3.4 Correcting

In some cases, LLMs may generate syntactically incorrect queries. These queries are clear candidates for corrections because they do not provide the correct answer. To solve this problem, we apply an LLM-based query correction tool that utilizes a self-reflective approach(Shinn et al., 2023). We use a small number of examples to guide the correction program, helping it learn from both correct and incorrect previously generated queries. The results of the Divide-and-Conquer and In-domain Knowledge are fed into the query repairer, which combines LLM's own knowledge to correct syntax or logic errors. Of course the final result is also corrected after the third Candidate Agent step, and details were shown in Appendix's Figure 4. We find that the type and number of examples affect the correction results, and we will analyze their impact in detail in the next section.

Methods	Model	EX(%)	EM(%)
DIN-SQL	GPT-4	74.2	60.1
PICARD	T5-3B	75.1	71.9
GRAPHIX	T5-3B	78.2	75.6
GRAPHIX+PICARD	T5-3B	79.3	77.1
Self-Debugging	code-davinci-002	84.1	77.1
DAIL-SQL	GPT-4	86.2	-
DPG-SQL	GPT-4	85.6	-
DAIL-SQL	GPT-4	86.6	70.7
DFM-SQL(ours)	Qwen2.5	85.3	**85.6**

Table 2: Execution accuracy (EX) and exact set match accuracy (EM) on the holdout test set of Spider

4 Experiments

4.1 Baseline Models

In the experiments, since the base model needed to demonstrate strong capabilities in mathematical, symbolic, and logical reasoning, we conducted zero-shot inference tests to evaluate multiple candidate models. Detailed experimental records for each model are provided in Appendix's Table 7. Ultimately, we selected the advanced Qwen2.5-Coder as the pre-trained model due to its superior performance(Qwen et al., 2025), and the related parameters is shown in Appendix's Table 6. In the process of constructing an in-domain knowledge base, we use Qwen2.5-Instruct, which makes its database have excellent and large internal knowledge. In the correction phase, we use leading LLMs in code or symbolic reasoning, such as DeepSeek and Qwen2.5-Instruct.

4.2 Dataset

Spider contains 10,181 questions and 5,693 unique complex SQL queries across 200 databases, covering 138 domains, each with multiple tables. The standard protocol for this dataset divides it into 8,659 training examples across 146 databases, 1,034 development examples across 20 databases, and 2,147 test examples across 34 databases. The databases used in these collections do not overlap. Since language models without access to database content often face schema linking challenges, our hints for the Spider dataset include sample rows from each table to assist the model in schema linking. Additionally, we link the provided knowledge of each field as hints, placed immediately after each question. However, due to constraints like limited context window size, available field knowledge, and sample row inclusion, we had to reduce the number of presentations in the dataset prompts.

Methods	Correcting	EX(%)	EM(%)
Baseline	×	71.3	66.8
In-domain Knowledge	×	72.7↑1.4	69.7↑2.9
Divide-and-Conquer	×	77.9↑6.6	74.6↑7.8
Baseline	✓	76.5↑5.2	73.8↑7.0
In-domain Knowledge	✓	77.7↑6.4	75.8↑9.0
Divide-and-Conquer	✓	81.6↑10.0	80.1↑13.3
Candidate Agent	×	82.8↑11.5	83.5↑16.7
Candidate Agent	✓	**84.3↑13.0**	**85.3↑18.5**

Table 3: Performance of Multiple Agent Integration compared with Baselin

4.3 Evaluation Metrics

We use Execution Accuracy(Qin et al., 2022b) as the evaluation metric for all experiments, calculating the proportion of correct execution SQL query results in the dataset, which reflects the percentage of predictions matching the golden SQL queries execution results. We also use Execution Match(Qin et al., 2022b) as an evaluation metric to measure how well the model-generated SQL queries matches the golden SQL queries, calculating the percentage of predictions that correctly align with the golden SQL query results.

4.4 Results

We evaluated the generalizability of the proposed DFM-SQL by conducting an end-to-end assessment on the Spider test set, without modifying the small sample size in the cue or training a new selection model, meaning no data from the target distribution was used. This approach enables us to test DFM-SQL's performance on unknown queries and database distributions, in contrast to data from the training distribution. Table 2 shows that DFM-SQL achieves 84.3% execution accuracy and 85.3% precise matching accuracy on the Spider test set, ranking second in execution accuracy and first in precise matching accuracy among methods specifically trained or cued to optimize for the Spider dataset. This highlights the strong versatility of DFM-SQL and its ability to generate high-quality Text-to-SQL for unknown samples from diverse distributions and unique challenges.

domain Knowledge + Divide-and-Conquer To validate the complex logic processing after knowledge supplementation of In-domain Knowledge data tables and Divide-and-Conquer disassembly, we conducted ablation experiments in a more realistic scenario, using direct sample-less corrections without the Candidate Agent, as shown in Table 3. We compared the performance of In-domain Knowledge and Divide-and-Conquer in generating individual candidate queries versus raw Spider hints as a baseline for evaluating the quality of hints. The experimental results show that in In-domain Knowledge, the constructed in-domain knowledge base significantly improves generation performance, boosting the Execute Match and Exact Match metrics by about 2-3 percentage points each. This result demonstrates In-domain Knowledge's ability to generate high-quality synthesized examples by understanding structured knowledge, effectively enhancing the performance of Large Language Models. After splitting the complex problem, the final SQL queries generated by sub-SQL queries merging outperforms both the In-domain Knowledge method and the Baseline (with EX reaching 77.9 and EM reaching 74.6). This indicates that LLMs excel at chain-of-thought reasoning, understanding the problem, and generating high-quality candidate SQL queries. Our proposed approach significantly improves SQL queries generation performance and helps us achieve our goal of generating high-quality candidates while maintaining diversity. Additionally, Correcting proves its importance by enhancing the quality of the candidate pool SQL queries and boosting the performance of all candidate generators by nearly 4%.

Candidate Agent + Correcting We analyze the binary selection accuracy of the selection agent in pairwise comparisons, where one candidate is correct and the other is incorrect. For the correct candidate SQL queries that can be executed accurately, we prioritize its selection directly. For incorrect SQL queries, which have a very high number of error factors, we used the pre-trained LLM of the correct SQL for scoring, and filtered the SQL queries with high scores as the final candidate SQL queries. To evaluate the potential of efficiently selecting the correct SQL queries from a candidate pool, we applied the Divide-and-Conquer and In-domain Knowledge methods to all samples in the Spider development set. For each method, we generated m candidate SQL queries (totaling 2m queries). We then combined the highest-scoring and correctly executed SQL queries into a final group for minimal correction. After selecting and refining high-quality candidate SQL queries with diverse characteristics, we found that an ensemble approach is highly effective for extracting and leveraging this knowledge.

Methods	Correcting	Metric	Easy	Medium	Hard	Extra-hard	All
IdK		EX	86.6	78.8	59.8	56.6	72.7
		EM	87.7	77.6	56.8	44.0	69.7
DaC		EX	88.7	82.1	70.6	66.1	78.4
		EM	90.4	83.0	66.5	51.5	75.8
IdK + DaC		EX	89.4	82.1	69.8	63.9	78.0
		EM	91.5	82.0	64.1	47.6	74.5
IdK	✓	EX	84.9	80.2	66.7	63.0	75.5
		EM	86.8	79.2	64.8	50.1	72.9
DaC	✓	EX	89.1	85.1	76.2	70.3	81.6
		EM	91.5	86.9	73.9	56.6	80.1
IdK + DaC	✓	EX	88.9	86.0	74.7	71.4	81.8
		EM	91.1	86.3	71.9	56.9	79.6

Table 4: Performance on three methods. IDK for the method In-domain Knowledge, DaC for the Divide-and-Conquer, and ✓ is that experiment is added Correcting method

Samples	Models	Metric	Easy	Medium	Hard	Extra-hard	All
3	Deepseek-V3	EX	89.1	85.6	70.9	82.2	82.2
		EM	91.1	87.3	73.7	57.4	80.2
4	Deepseek-V3	EX	88.9	86.0	74.7	71.4	81.8
		EM	91.1	86.9	71.9	56.6	79.6
5	Deepseek-V3	EX	89.1	85.9	76.7	71.4	82.2
		EM	91.1	87.3	73.7	57.4	80.2
7	Deepseek-V3	EX	89.1	86.1	76.2	70.9	82.1
		EM	91.1	87.5	72.8	55.7	79.8
5	Qwen2.5-Coder	EX	89.1	86.1	76.2	70.9	82.1
		EM	91.1	87.5	72.8	55.7	79.8
5	GPT-4o	EX	89.1	86.1	76.2	70.9	82.1
		EM	91.1	87.5	72.8	55.7	79.8

Table 5: Performance of different difficulty levels with samples and models.

5 Analysis and Discussion

5.1 Comparison of Methods

Table 5 shows the performance of the In-domain Knowledge prompt, the Divide-and-Conquer prompt, and the In-domain Knowledge and Divide-and-Conquer prompt on the Spider at four levels of difficulty. As expected, the Divide-and-Conquer hints performed better on tasks above medium difficulty.The Divide-and-Conquer hints performed better than the In-domain Knowledge hints, showing that the model is more skilled at reasoning and decomposing subproblems, improving its overall understanding and analysis. This also poses a greater challenge in developing more effective In-domain Knowledge methods.

5.2 Few-shot Correcting Comparison of Samples

Table 5 shows the effect of different number of samples on the error correction ability under the condition of using Divide-and-Conquer hints. We find that as the number of samples gradually increases, LLMs are able to learn more correct and incorrect features, and the more effective it is for candidate queries error correction. However, we found that the number of samples also affects the length of the context. If there are too many samples, the more features its LLM needs to memorize, which may mislead the LLM to correct the candidate SQL queries. It is verified that when $K = 5$ will make the error correction performance of LLMs better, especially we use Deepseek-V3 to correct SQL queries.

6 Conclusion

We propose DFM-SQL, an innovative framework that generates diverse, high-quality SQL queries and precisely identifies the optimal query during test-time computation. This framework combines an in-domain knowledge base, a chain-of-thought hinting approach, a hint correction method, and a pairwise comparison mechanism to accurately evaluate candidate statement quality. DFM-SQL sets new benchmarks in text-to-SQL tasks, highlighting the effectiveness of test-time computation in producing diverse queries and identifying the best responses. DFM-SQL tackles the key issues of query diversity and selection optimization, paving the way for advancements in complex reasoning tasks for practical use.

7 Limitations

7.1 Limitations

DFM-SQL integrates multiple large language models (LLMs) to generate candidate SQL queries and combines a divide-and-conquer strategy with a domain-specific knowledge base, significantly enhancing the quality and diversity of the generated SQL. However, the introduction of multi-model integration and complex strategies also results in higher computational resource consumption, which may limit its practical application in resource-constrained environments. Future research could explore model compression, knowledge distillation techniques, or more efficient inference methods to reduce computational costs and improve the framework's practicality. Additionally, DFM-SQL employs few-shot learning to fine-tune and optimize the generated SQL queries, but its effectiveness heavily relies on the representativeness and domain relevance of the example data. If the example data significantly differs from the target database's domain, the performance of few-shot learning may degrade considerably.

Although DFM-SQL achieves an impressive execution accuracy of 85.3% and an exact match accuracy of 86.3% on the Spider dataset, with only a 1% gap between the two metrics, this consistency may be limited to specific datasets and task settings. In the future, we plan to migrate the work to other datasets for further validation. In other datasets or more complex query scenarios, the framework's generalizability and robustness still require further verification. To address this, external knowledge bases or domain expert input could be incorporated to enhance the coverage and accuracy of domain knowledge, thereby further improving the framework's adaptability and performance.

7.2 Ethical Consideration

The DFM-SQL framework must strictly adhere to data privacy protection principles when processing databases and generating SQL queries. Any data involving personal privacy or sensitive information should be anonymized or desensitized before use to ensure that the privacy rights of data subjects are not violated. Researchers, developers, and users of the DFM-SQL framework are responsible for its application scenarios and outcomes. If the generated SQL queries lead to data leaks, incorrect decisions, or other negative consequences, the relevant parties should take prompt measures and assume corresponding responsibilities.

References

Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. SADGA: Structure-aware dual graph aggregation network for text-to-SQL. In *Advances in Neural Information Processing Systems*.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2541–2555, Online. Association for Computational Linguistics.

Zhibo Chu, Zichong Wang, and Qitao Qin. 2024. Leveraging prior experience: An expandable auxiliary knowledge base for text-to-SQL.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot text-to-sql with chatgpt. *Preprint*, arXiv:2307.07306.

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.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. 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, 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. 2023. 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. *arXiv preprint arXiv:2303.13547*.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? text-to-SQL in the age of well-reasoned language models. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Gyutae Park, Seojin Hwang, and Hwanhee Lee. 2024. Low-resource cross-lingual summarization through few-shot learning with large language models. In *Proceedings of the Seventh Workshop on Technologies for Machine Translation of Low-Resource Languages (LoResMT 2024)*, pages 57–63, Bangkok, Thailand. Association for Computational Linguistics.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *The Thirteenth International Conference on Learning Representations*.

Mohammadreza Pourreza and Davood Rafiei. 2023a. 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. 2023b. Evaluating cross-domain text-to-SQL models and benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1601–1611, Singapore. Association for Computational Linguistics.

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed text-to-SQL with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA. Association for Computational Linguistics.

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, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022a. A survey on text-to-sql parsing: Concepts, methods, and future directions. *Preprint*, arXiv:2208.13629.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022b. A survey on text-to-sql parsing: Concepts, methods, and future directions. *ArXiv*, abs/2208.13629.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA. MIT Press.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *Preprint*, arXiv:2405.16755.

Zhao Tan, Xiping Liu, Qing Shu, Xi Li, Changxuan Wan, Dexi Liu, Qizhi Wan, and Guoqiong Liao. 2024. Enhancing text-to-SQL capabilities of large language models through tailored promptings. In *Proceedings* of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), pages 6091–6109, Torino, Italia. ELRA and ICCL.

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.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Eric Sun, and Yue Zhang. 2023. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *ArXiv*, abs/2312.02003.

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.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

Danna Zheng, Mirella Lapata, and Jeff Pan. 2024. Archer: A human-labeled text-to-SQL dataset with arithmetic, commonsense and hypothetical reasoning. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 94–111, St. Julian's, Malta. Association for Computational Linguistics.

A Appendix

setting	parameter
train_epochs	5
learning_rate	5e-5
cutoff_len	2048
Batch_size	2
Optimizer	AdamW
Warm_up_ratio	0.05
Lr_scheduler	Cosine
Gradient_accumulation_steps	8
LoRA_Rank	128
LoRA_Alpha	512

Table 6: Pretrained Models Parameters and LoRA Parameters

In-domain Knowledge Base Prompt

system

You are now an expert in SQL statements. I will give you information about a database. You need to describe the information about the database, including the database, the tables that corresponds to that library, and the columns that correspond to that table. And return the description information to me in the format. I will give you an example.\n
Example\n
First: perpetrator database is about information of perpetrator.\n
Second: people table records basic information about the person, and People_ID column records the person's identity ID...\n
Third: perpetrator table records basic information about the crime, Perpetrator_ID column records the crime number, and Perpetrator_ID is a primary key, and People_ID is a foreign key intended to link people table...\n
In your response, you do not need to mention your intermediate steps. Do not include any comments in your response. Do not need to start with the special symbol.\n

user

The database information is as follows:\n
First: perpetrator database has perpetrator, people tables.\n
Second: below are columns of table, primary_keys of table:\n1
The perpetrator table has Perpetrator_ID, People_ID, Date, Year, Location, Country, Killed, Injured columns, and Perpetrator_ID is primary_key.\n2
The people table has People_ID, Name, Height, Weight, Home Town columns, and People_ID is primary_key.\n
Third: below are foreign_keys of tables:\nperpetrator.People_ID to people.People_ID.\n"

assistant

First: perpetrator database is about information of perpetrators and people. \n
Second: perpetrator table records basic information about the crimes, and Perpetrator_ID column records the crime number, and Perpetrator_ID is a primary key, and People_ID column is a foreign key intended to link the people table... \n
Third: people table records basic information about the person, and People_ID column records the person's identity ID, and People_ID is a primary key...,

db_id

perpetrator

Figure 2: An example of the In-domain Knowledge method

Divide-and-Conquery Prompt

system

You are now an expert in SQL statements. I will give you information about a database and subsequently ask you a question. You need to help me parse the question following the steps I gave you and answer a final SQL statement.\n
Question:xxx
First. It is divided and conquered:\n
Main Question{asked question}:xxx
Analysis{The purpose is to break down the question,and to complete an analysis}:xxx
Pseudo SQL{The purpose is to give a predicted SQL}:xxx
Divided-question 1{The purpose is to isaggregated the main question}:xxx
Analysis{The purpose is to perform an analysis based on the disassembled problem}:xxx
Pseudo SQL{The purpose is to give a predicted SQL}:xxx
If the problem and SQL are more complex, you can break it down further....
Divided-question 1.1:xxx Analysis:xxx Pseudo SQL:xxx...\n
Second. Assembling SQL{You need to merge the above disassembled SQL and have it executable.}:xxx
Divided-question 2{}:xxx more..
Main Question(it is aim to combine the isaggregated SQLs):xxx \n
Third. Simplification and Optimization:
Final Optimized SQL Query:xxx \n
Optimized SQL{best SQL}:xxx

user

The database information is as follows:\n
First: perpetrator database has perpetrator, people tables.\n
Second: below are columns of table, primary_keys of table:\n1
The perpetrator table has Perpetrator_ID, People_ID, ... and Perpetrator_ID is primary_key...\n2
The people table has People_ID, Name, Height...\n
Third: below are foreign_keys of tables:\nperpetrator.People_ID to people.People_ID.\n"

assistant

SELECT count(*) FROM head WHERE age > 56.

db_id

perpetrator

Figure 3: An example of the Divide-and-Conquer method

Correcting Prompt

system

You are now an expert in fixing SQL and the SQL I have given you is wrong. The types of errors are logical errors, table join errors, special word errors (not in, except, group by improperly used), column name and table name error, having clause errors etc. I will now give you an incorrect SQL along with data information and query issues, please help me to fix it correctly and provide me the correct SQL.\n
I'll give you some sample corrections: \n
ques1:What are the names of clubs, ordered descending by the average earnings of players within each?\n
wrong sql1:SELECT T1.Name FROM club AS T1 JOIN player AS T2 ON T1.Club_ID...\n
correct sql1:SELECT T1.Name FROM club AS T1 JOIN player AS T2 ON T1.Club_ID...\n
ques2:What are the first names of all buyers and what products did they buy? List them in pairs.\n
wrong sql2:SELECT T3.customer_first_name , T2.product_name FROM orders AS T1 JOIN order_items...\n
correct sql2:SELECT T1.customer_first_name , T4.product_name FROM Customers AS T1 JOIN Orders...\n
ques3:List the order date of the orders who are placed by customers with at least 2 payment methods.\n
wrong sql3:SELECT T1.date_order_placed FROM orders AS T1 JOIN customers AS T2 ON T1.customer_id = T2.customer_id WHERE T2.customer_id...\n
correct sql3:SELECT date_order_placed FROM Orders WHERE customer_id IN (SELECT T1.customer_id FROM Customers AS T1 JOIN Customer_Payment_Methods A...\n
In your response, you do not need to mention your intermediate steps. Do not include any comments in your response. Do ont include line break. Do not need to start with the Special symbol. Your fix answer should be concise and efficient.

user

The database information is as follows:\n
First: perpetrator database has perpetrator, people tables.\n
Second: below are columns of table, primary_keys of table:\n1
The perpetrator table has Perpetrator_ID, People_ID, ... and Perpetrator_ID is primary_key...\n2
The people table has People_ID, Name, Height...\n
Third: below are foreign_keys of tables:\nperpetrator.People_ID to people.People_ID.\n"

assistant

SELECT count(*) FROM head WHERE age > 56.

db_id

perpetrator

Figure 4: An example of the Correcting method with three samples

Models	Metric	Easy	Medium	Hard	Extra-hard	All
Qwen2.5-Coder-14B	EX	**80.0**	45.6	41.5	22.4	**48.4**
	EM	78.1	39.7	35.2	12.6	**42.6**
Qwen2.5-14B-Instruct	EX	79.1	44.1	37.2	28.9	45.1
	EM	75.1	32.5	31.8	11.2	23.8
Internlm2.5-8B	EX	67.1	34.0	21.7	11.4	32.8
	EM	74.1	30.2	31.3	10.2	21.8
CodeLlama-13B	EX	68.3	45.6	43.4	29.7	47.5
	EM	64.7	34.0	29.4	10.9	35.9
CodeFuse-13B	EX	69.1	42.2	36.2	28.9	40.1
	EM	61.1	33.5	28.8	10.7	34.1

Table 7: Preformance on different large language models.