Enhancing Text-to-SQL with Question Classification and Multi-Agent Collaboration

Zhihui Shao[1], Shubin Cai[1,2*], Rongsheng Lin[1], Zhong Ming[2,3]
[1]College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
[2]Laboratory of Artificial Intelligence and Digital Economy (Shenzhen), Shenzhen, China
[3]College of Big Data and Internet, Shenzhen Technology University, Shenzhen, China
{shaozhihui2022,linrongsheng2022}@email.szu.edu.cn
{shubin,mingz}@szu.edu.cn

Abstract

Large Language Models (LLMs) have recently demonstrated remarkable performance in Text-to-SQL tasks. However, existing research primarily focuses on the optimization of prompts and improvements in workflow, with few studies delving into the exploration of the questions. In this paper, we propose a Text-to-SQL framework based on question classification and multi-agent collaboration (QCMA-SQL). Specifically, we first employ multiple cross-attention mechanisms to train a schema selector to classify questions and select the most suitable database schema. Subsequently, we employ the appropriate agents based on the varying difficulty levels of the questions to generate preliminary SQL queries. Moreover, we implement syntax validation and execution optimization steps to generate final SQL queries. Experimental results on the Spider dataset show that the QCMA-SQL framework achieves an execution accuracy of 87.4%, outperforming state-of-the-art methods. Through ablation studies, we find that classifying the questions ultimately leads to a 2.8% increase in execution accuracy.

1 Introduction

The Text-to-SQL task converts natural language queries to Structured Query Language (SQL), enabling users to easily retrieve database information without SQL knowledge (Wang et al., 2022).The primary challenges of the Text-to-SQL task originate from the comprehensive understanding required for natural language question intentions, and the complexity involved in precisely mapping entities in natural language to tables and column names within database schemas (Li et al., 2023a). As technology evolves and research progresses, the academic community has proposed various solution strategies. Currently, mainstream methods are mainly divided into two categories: one is based on pre-trained language models (PLMs) (Li et al., 2023a; Scholak et al., 2021), and the other relies on advanced methods supported by LLMs (Hong et al., 2024; Wang et al., 2023; Rajkumar et al., 2022).

With the continuous progress in the field of deep learning, PLMs have achieved remarkable success in various tasks of natural language processing (Li et al., 2023b). RAT-SQL (Wang et al., 2020), strengthens the SQL encoder's capability in schema encoding, relation, and feature expression by connecting primary and foreign key relations in data tables through a graph model. The RESDSQL model (Li et al., 2023a), by decoupling schema alignment and query skeleton analysis, enhances the precision and robustness of SQL parsing. However, due to the inherent limitations in the parameter scale of PLMs, traditional methods based on deep learning are no longer the most ideal solution for the Text-to-SQL task.

Recent research outcomes demonstrate that LLMs exhibit outstanding performance in Text-to-SQL tasks, an achievement attributable to LLMs' significant advantages in natural language understanding and reasoning capabilities (Liu et al., 2023; Rajkumar et al., 2022; Sun et al., 2023). In practice, the vast majority of approaches employ meticulously designed prompts to tap into the potential of LLMs in Text-to-SQL transformation tasks, including the use of thought chains and task decomposition strategies to construct complex prompts (Tai et al., 2023). For instance, C3-SQL (Dong et al., 2023) introduces bias hints that are calibrated using a self-consistent policy designed for the Spider dataset to guide GPT-4 (Achiam et al., 2023). Concurrently, DIN-SQL (Pourreza and Rafiei, 2024) subdivides the text-to-SQL task into a series of smaller sub-tasks, customizing distinct directive prompts for each sub-task to steer GPT-4 to complete each sub-task and construct the final SQL query incrementally. DAIL-SQL (Gao

[*]Corresponding author.

Findings of the Association for Computational Linguistics: NAACL 2025, pages 4340–4349
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

et al., 2024) adopts an approach that encodes the structural knowledge within SQL queries, selecting a small sample set based on skeleton similarity for demonstration. Nevertheless, the methods above still face challenges when dealing with large-scale databases, complex user questions, and erroneous SQL outputs. Firstly, there is a lack of more comprehensive consideration for questions with different levels of difficulty. Secondly, schema linking using LLMs incurs expensive costs.

To address these issues, this paper introduces QCMA-SQL, an innovative Text-to-SQL framework that integrates traditional PLMs with state-of-the-art LLMs to propose a novel methodology. The framework decomposes the Text-to-SQL task into two principal stages: question classification and SQL generation. During the question classification stage, a schema selector classifies the question and chooses the most relevant schema items to the question from a large number of database schemas. In the SQL generation stage, we input questions of different difficulty to different agents to generate initial SQL queries. Other agents then perform error detection and execution optimization.

To assess the efficacy of our framework, we conducted experiments on the Spider (Yu et al., 2018), Spider-DK (Gan et al., 2021b), Spider-SYN (Gan et al., 2021a), Spider-Realistic (Deng et al., 2021), and CSpider (Min et al., 2019). Our method achieved an execution accuracy of 87.4% on the Spider dataset and an exact match accuracy of 69.5% on CSpider, showcasing the effectiveness and robustness of our approach across various datasets.

Our main contributions and results are summarized as follows: (1) A Text-to-SQL framework, called QCMA-SQL, is innovatively proposed for high-precision and high-efficiency SQL generation. (2) Considering the importance of question classification and schema linking, this study introduces a schema selector based on multiple cross-attention mechanisms to enhance the accuracy of classification. (3) Through extensive experimental validations, QCMA-SQL achieved new state-of-the-art (SOTA) performance levels on Spider.

2 Related Work

Inspired by pre-training techniques, seq2seq models such as T5 (Raffel et al., 2020) and BART (Lewis et al., 2020) have begun to demonstrate superior performance on this task, indicating a research trend toward fine-tuning large-scale PLMs on specific database-related data. RESDSQL (Li et al., 2023a) simplifies the process of converting natural language into SQL queries by decoupling schema matching and query skeleton analysis and employs a hierarchical enhanced encoder and a skeleton-aware decoder to improve the accuracy and robustness of SQL parsing. Nevertheless, PLM-based approaches are far from the standard for reliable text-to-SQL parsers.

With the rise of LLMs such as GPT-4 (Achiam et al., 2023), Codex (Chen et al., 2021), PaLM (Chowdhery et al., 2023) and StarCoder (Li et al., 2023c), researchers have proposed diverse strategies to exploit the advanced reasoning abilities of these models, focusing on creating superior prompts for probing SQL generation potential. For instance, Chang (Chang and Fosler-Lussier, 2023) have delved into mind-chain prompt strategies to enhance the LLM reasoning capacity, while models such as DAIL-SQL (Gao et al., 2024) and C3-SQL (Dong et al., 2023) optimize the execution process and outcomes through precise calibration of model inputs, query classification and decomposition, and special logical handling tailored for the text-to-SQL tasks. MAC-SQL (Wang et al., 2023) proposes a multi-agent collaboration framework, in which multiple LLMs are responsible for simplifying database schemas, decomposing complex questions, and incrementally generating SQL. CodeS (Li et al., 2024) enhances SQL generation and natural language understanding capabilities through supervised fine-tuning and a carefully curated dataset, and achieves rapid domain adaptation through bidirectional data augmentation techniques.

3 Method

3.1 General Architecture

The QCMA-SQL framework consists of two core components: the schema selector and the multi-agent collaboration mechanism. This framework ingeniously integrates the strengths of Pre-trained Language Models and Large Language Models, significantly enhancing the accuracy of the Text-to-SQL task. Figure 1 illustrates the overall structure of the QCMA-SQL framework. Initially, the schema selector categorizes the question into simple or complex types and selects the most relevant database schema from them. Subsequently, we employ specific agents to generate preliminary SQL queries for questions of varying difficulty levels.

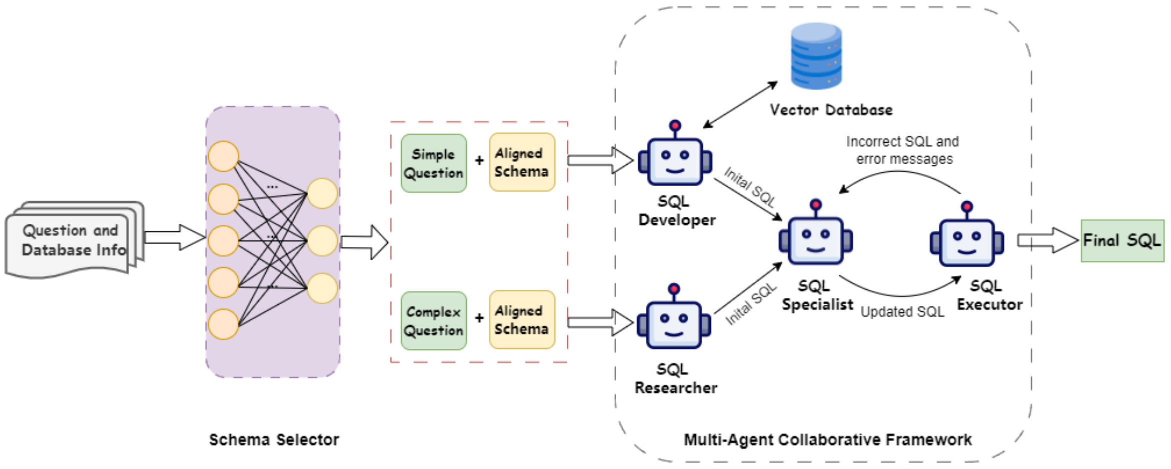


Figure 1: An overview of QCMA-SQL framework. We train a schema selector to classify questions and filter the schema items. Then we take the question, and the aligned schema sequence as the input of the multi-agent collaboration framework.

In the end, through the collaborative interaction among multiple agents, we deploy a multi-stage iterative correction mechanism. This mechanism integrates error detection and execution optimization, aiming to ensure the accuracy and reliability of SQL queries.

3.2 Schema Selector

The function of the schema selector is to categorize the difficulty of questions while choosing the most relevant database schema. The Spider dataset provides specific usage of components for each SQL query, such as "orderBy," "groupBy," "join," "having," and "limit," etc. We divide the training set into simple or complex categories, which depend on the number of SQL keywords used, the presence of nested subqueries, and the utilization of column selection or aggregation. During the training process, we employ multiple cross-attention mechanisms to effectively capture the intrinsic connections between questions, difficulty level, table names, and column names. Details of the schema selector are shown in Figure 2.

Encoder. We define the difficulty level, the table, and the column in the relational database as: $\mathbf{D} = \{d_1, d_2\}$, $\mathbf{T} = \{t_1, t_2, \cdots, t_N\}$, $\mathbf{C} = \{c_1^1, \cdots, c_{n1}^1, c_1^2, \cdots, c_{n2}^2, \cdots, c_1^n, \cdots, c_{nN}^n\}$.

$\mathbf{D}$ denotes the set of difficulty levels, $\mathbf{T}$ denotes the set of tables, $\mathbf{C}$ denotes the set of columns, N denotes the number of tables, and c_i^j denotes the i-th column of the j-th table. We concatenate questions, difficulty levels, and database information as the input to the encoder: $S = q \mid d_1, d_2 \mid t_1{:}c_1^1, \cdots, c_{n1}^1 \mid \cdots \mid t_N{:}c_1^N, \cdots, c_{nN}^N$.

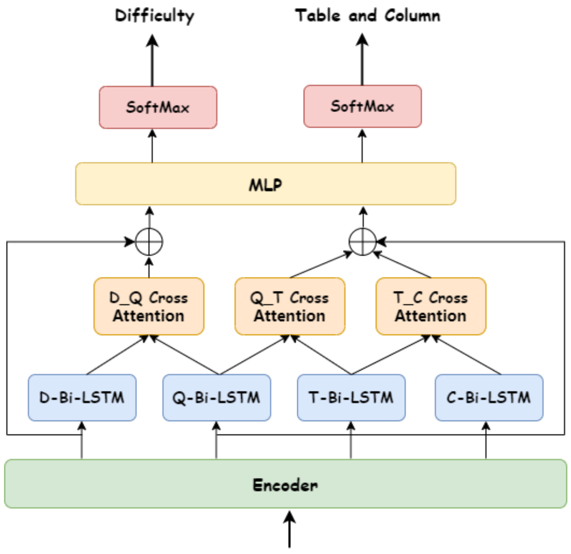


Figure 2: Schema selector. Multiple cross-attention mechanisms extract features between questions, difficulty levels, table names, and column names.

To extract more essential semantic features, we utilize four separate Bi-LSTM layers to pool difficulty levels, questions, tables, and columns sequence respectively. After pooling, the embedding of the question can be denoted by $\mathbb{Q} \in R^{1\times \mathrm{d}}$, the embedding of each table can be denoted by $T_i \in R^{1\times \mathrm{d}}(\mathrm{i} \in \{1, ..., \mathrm{N}\})$, and the embedding of each column can be denoted by $C_k^i \in R^{1\times \mathrm{d}}(\mathrm{i} \in \{1, ..., \mathrm{N}\}, \mathrm{k} \in \{1, ..., \mathrm{n_i}\})$.

Multiple Cross-Attention Mechanisms. We notice that some questions only mention column names but not table names, which could affect the accuracy of schema selection. Thus, we propose a method with multiple cross-attention mechanisms to assist the model in difficulty classification and schema selection. Specifically, we employ a difficulty-question cross-attention layer to embed difficulty information into the semantic context of the question. Additionally, we utilize a question-table cross-attention layer and a table-column cross-attention layer to embed table information and column information into the semantic context of the question.

$$Q^D = MultiHeadAttn(Q, D_i, D_i, \mathrm{h}) \qquad (1)$$

$$Q^T = MultiHeadAttn(Q, T_i, T_i, \mathrm{h}) \qquad (2)$$

$$T_i^c = MultiHeadAttn(T_i, C_k^i, C_k^i, \mathrm{h}) \qquad (3)$$

Here, D_i represents the difficulty levels, T_i represents the table name, and C_j^k represents the column name. Q represents the question, and h denotes the number of heads in multi-head attention.

Loss Function. Given that SQL queries typically involve only a small number of tables and columns in the database, the label distribution in our training set is highly imbalanced. To mitigate this issue, we adopt a balanced cross-entropy loss function for our classification loss. We construct the loss function for the schema selector in a multi-task learning way, including the loss for difficulty classification, the loss for table classification, and the loss for column classification.

3.3 Multi-Agent Collaborative Framework

Following the initial phase, we categorize the questions into two types and select the most relevant database schema. Leveraging the exceptional natural language understanding and advanced reasoning capabilities of LLMs, We develop a multi-agent collaborative SQL generation framework. As depicted in Figure 1, the framework is composed of four agents: SQL Developer, SQL Researcher, SQL Executor, and SQL Specialist. We meticulously design prompts for each agent to maximize their respective strengths. For simple questions, we utilize the SQL Developer, which employs retrieval enhancement technology to recall similar question-SQL pairs from the vector database as contextual information. For complex questions, we apply the SQL Researcher, which is based on task decomposition and chain-of-thought(CoT) to generate preliminary SQL queries. These SQL are then syntax-checked and executed by the SQL Executor. If the execution is successful, the output is the final SQL query; if not, the SQL Specialist modifies the SQL query based on the error information and resubmits it to the SQL Executor for execution and verification.

The implementation of LLM-based text-to-SQL process to generate executable SQL query Y is formulated as:

$$Y = LLM(\mathcal{Q}, \mathcal{S}, \mathcal{I} \mid \theta) \qquad (4)$$

where $\mathcal{Q}$ represents the user question. $\mathcal{S}$ is the aligned database schema, and $\mathcal{I}$ represents the instruction for the Text-to-SQL task, which performs indicative guidance to guide the LLMs for generating an accurate SQL query. $LLM(\cdot \mid \theta)$ is a LLM with parameter θ.

SQL Developer. The SQL Developer excels at addressing simple questions, which typically require only a single step of reasoning. We store the training set of question-SQL pairs and Data Definition Language (DDL) in a vector database, enabling the retrieval of the most similar cases to the given question. This flexible strategy provides the agent with more precise cases, maximizing the potential of LLMs in few-shot learning environments.

SQL Researcher. The SQL Researcher is adept at handling complex questions. For such questions, we employ a method of question decomposition combined with CoT. Initially, we break down the complex question into multiple sub-questions that are in a progressive relationship. We then generate SQL queries for each sub-question individually, step by step arriving at the complete SQL.

SQL Executor. The SQL Executor is responsible for receiving SQL queries generated by either the SQL Developer or the SQL Specialist and conducting preliminary checks for syntax accuracy. Upon confirming that there are no apparent syntax errors, the SQL Executor proceeds to execute the SQL query. In case of successful implementation, the executed SQL query is defined as the final output. However, if errors occur during the execution process, the system invokes assistance from the SQL Specialist to conduct a profound analysis.

SQL Specialist. The expertise of the SQL Specialist lies in revising SQL queries based on the feedback from error messages. Taking into consideration the database schema, the flawed SQL queries, and the corresponding error messages, the

Dataset	Table	Column	Difficulty
Spider	**0.9984**	**0.9973**	**0.9893**
CSpider	0.9930	0.9904	-

Table 1: Table, column, and difficulty AUC scores for the trained schema selector.

SQL Specialist reconstructs the SQL queries accordingly. These revised queries are then resubmitted to the SQL Executor for further error checking and verification.

4 Experiments

Datasets. We conduct extensive experiments on the cross-domain large-scale Text-to-SQL benchmark: Spider (Yu et al., 2018) and CSpider (Min et al., 2019). We also assess our models' robustness across three more challenging benchmarks: Spider-DK(Gan et al., 2021b), Spider-SYN(Gan et al., 2021a), and Spider-Realistic(Deng et al., 2021). The Spider dataset is commonly utilized for evaluating text-to-SQL parsing performance across multiple databases. This dataset comprises 7,000 Question-SQL pairs in the training set and 1,034 in the development set, covering 200 distinct databases and 138 domains. CSpider is a large, complex cross-domain semantic parsing and Text-to-SQL dataset in Chinese, translated from Spider. Spider-DK, Spider-Syn, and Spider-Realistic are variants derived from the original Spider dataset. They are designed to mimic questions that users might pose in real-world scenarios.

Evaluation Metrics. We consider two evaluation metrics—Execution Accuracy (EX) and Exact Match Accuracy (EM). EX quantifies the percentage of evaluation questions for which the inferred queries produce identical execution outcomes to the ground truth queries, against the aggregate query count. EM gauges the accuracy of predicted SQL sub-clauses by examining each sub-clause as an individual set and ensuring complete correspondence with the reference query's sub-clauses, deeming a SQL prediction accurate only if all elements align with the true query components.

Baselines. Our benchmark study evaluates the performance of two types of Text-to-SQL methods. **PLM-based methods:** T5-3B + PICARD (Scholak et al., 2021), RASAT (Qi et al., 2022) and RES-DSQL (Li et al., 2023a). **LLM-based methods:** GPT-4 (Achiam et al., 2023), SQL-PaLM (Sun et al., 2023), DIN-SQL (Pourreza and Rafiei, 2024), DAIL-SQL (Gao et al., 2024), RAG+SP&DRC (Guo et al., 2023), Codes (Li et al., 2024) and MAC-SQL(Wang et al., 2025).

Method	Model	EX
PLM-based	T5-3B + PICARD	79.3
	RASAT	80.5
	RESDSQL	84.1
LLM-based	GPT-4 (zero-shot)	72.9
	GPT-4 (few-shot)	76.8
	Fine-tuned SQL-PaLM	82.8
	DIN-SQL	82.8
	DAIL-SQL	84.4
	RAG+SP&DRC	85.0
	CodeS-7B	85.4
	MAC-SQL	86.7
Ours	QCMA-SQL	**87.4**

Table 2: Execution accuracy on Spider dev (%).

Implementation Details. We run all of our experiments on NVIDIA V100 GPU with 32GB memory. In the first phase, we employ DeBERTa-v3 as the encoder, and the number of heads h in the dual attention mechanism is set to 8. The optimization process utilizes the AdamW optimizer, with a batch size of 8, and a learning rate set at 2e-5. In the second phase, we use DeepSeek-V3 as the agent and design specific prompts for different agents. Specifically, we adopt a 3-shot setting in the prompts of the SQL Developer. We set the temperature of each agent to 0 to actively utilize the model's determinism features. In addition, we use Chroma as the vector database.

4.1 Main Results

In Table 1, we present the performance of the schema selector across different datasets. The AUC metric is adopted to evaluate the models' classification accuracy. According to the experimental results, the schema selector achieved an AUC score of 0.9984 for table classification and 0.9973 for column classification on the Spider dataset, both of which exceed the performance metrics of the CSpider dataset. We speculate that this is because DeBERTa is more proficient in English. Furthermore, for the classification of question difficulty, the difficulty AUC score reached 0.9893. This indicates that the schema selector can accurately classify questions and link schema.

Table 2 shows the comparative performance of QCMA-SQL against other benchmark methods on

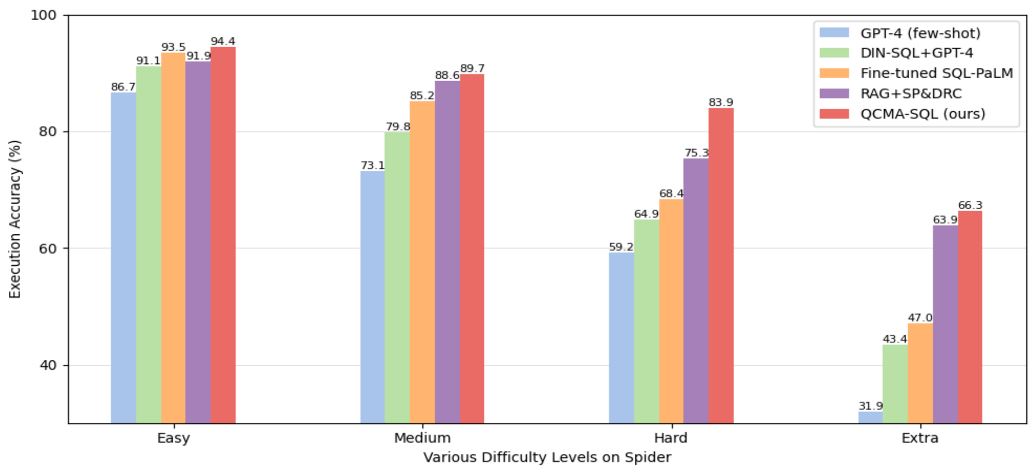


Figure 3: Test-suite accuracy at various difficulty levels on Spider.

Method	EM(%)
RAT-SQL	59.7
LGESQL + GTL + Electra + QT	64.0
LGESQL + ELECTRA + QT	64.5
RESDSQL	66.3
QCMA-SQL(ours)	**69.5**

Table 3: Exact Match Accuracy on CSpider.

Method	S-Syn	S-R	S-DK
T5-3B + PICARD	69.8	71.4	62.5
RASAT	70.7	71.9	63.9
RESDSQL	76.9	81.9	66.0
ChatGPT	58.6	63.4	62.6
Fine-tuned SQL-PaLM	70.9	77.4	67.5
RAG+SP&DRC	81.4	-	**81.1**
CodeS-7B	76.9	82.9	72.0
CodeS-15B	77.0	83.1	70.7
QCMA-SQL (ours)	**83.8**	**86.4**	80.7

Table 4: Evaluation of our method on Spider variants. S-Syn, S-R, and S-DK correspond to Spider-Syn, Spider-Realistic, and Spider-DK, respectively.

the Spider dev. The results indicate that our method outperforms all baselines in terms of execution accuracy. Specifically, the execution accuracy of QCMA-SQL reached 87.4%, which is 0.7% higher than the second-best method (MAC-SQL).

Table 3 presents a performance comparison between the method proposed in this study and several baseline methods on the CSpider dataset. QCMA-SQL outperforms the second-best method by 3.2%. The tabular names and column names in the CSpider dataset are in English, but the questions are in Chinese, increasing the challenge of mapping questions to database structures. The results demonstrate that our strategy performs best on the CSpider dataset. This fully validates the effectiveness and robustness of our method.

Evaluation on Robustness Benchmarks. Table 4 evaluates the robustness of QCMA-SQL on three Spider variants: Spider-DK, Spider-Syn, and Spider-Realistic. The experimental results show that QCMA-SQL exhibits outstanding performance compared to the best baseline. It achieves gains of 2.4% on Spider-Syn (from 81.4% to 83.8%), and 3.3% on Spider-Realistic (from 83.1% to 86.4%). These results demonstrate the model's good generalization on challenging real-world scenario datasets.

Various Difficulty Levels Analysis. As shown in Figure 3, these four difficulty levels are based on the official test suite provided by Spider. The results indicate that our model outperforms other models at all levels. It achieves gains of 0.9% on the easy level, 1.1% on the medium level respectively, 8.6% on the hard level, and 2.4% on the extra hard level. This is attributed to the multi-agent collaborative framework's ability to categorically handle questions of varying difficulty, suggesting that our model excels in processing SQL queries of arbitrary difficulty.

Tokens Consumption. The adoption of LLM for SQL generation in Text-to-SQL tasks is due to the significant performance advantages that LLM has exhibited over PLM. However, these optimization methods based on the LLM API consume a large number of tokens in the reasoning process. Experimental data shows that compared with ex-

Method	Easy	Medium	Hard	Extra	All
w/o SQL Researcher(QD and CoT)	**94.4**	**89.7**	73.6	62.0	83.6
w/o SQL Developer(RAG)	92.7	87.2	**83.9**	**66.3**	84.6

Table 5: Performance of the two algorithms at different difficulty levels (EX). For brevity, "Question decomposition " is denoted as "QD."

Method	EX
QCMA-SQL(ours)	**87.4**
w/o schema selector	84.2
w/o multi-agent collaborative	83.1

Table 6: Ablation study in Spider dev set

isting technologies, QCMA-SQL significantly reduces inference costs. Specifically, with the same use of the GPT-4 API, the DAIL-SQL(Gao et al., 2024) costs 323.2$ for a single round of experiments on Spider, while the corresponding cost for QCMA-SQL is only 60$.

4.2 Ablation Study

We conduct ablation studies on the Spider development set to evaluate the impact of two components: the schema selector and the multi-agent collaborative framework. The results are presented in Table 6.

Effect of Schema Selector. We investigate the influence of the schema selector on performance. When the schema selector is not used and all schema items from the database are input into the multi-agent collaborative framework, performance degradation occurs due to the inability of the LLM to accurately connect schema items because of the longer input sequence. Also, the lack of question classification leads to the inability to select the most appropriate agent to generate SQL. This highlights the importance of the schema selector.

Effect of Multi-Agent Collaborative Framework. Without the multi-agent collaborative framework, if the filtered schema items are input into a single Deepseek-V3 model, the lack of retrieval enhancement, error detection, and execution optimization functionalities leads to a decrease in the EX value. This demonstrates the indispensability of the multi-agent collaborative framework in the SQL generation process.

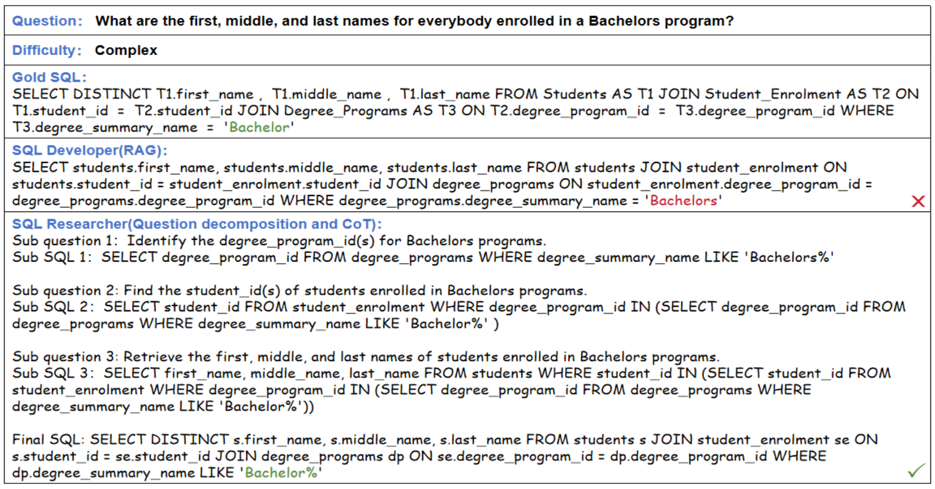
Effect of SQL Developer and SQL Researcher. We evaluate the algorithm performance improvement on different difficulty subsets. We do not categorize the questions but instead input them separately into SQL Developer, based on RAG, and SQL Researcher, based on question decomposition (QD) and CoT. As shown in Table 5, the strategy of question decomposition and CoT for complex questions effectively achieves step-by-step question resolution and improves performance, with an increase from 73.6% to 83.9% at the hard level and from 62.0% to 66.3% at the extra level. However, for simple questions, the method based on question decomposition and CoT decreases execution accuracy, from 94.4% to 92.7% at the easy level and from 89.7% to 87.2% at the medium level. We suggest that for simple questions solvable by a single step of reasoning, LLM tends to overly decompose them into multiple sub-questions. This approach not only unnecessarily increases the complexity of the questions but also introduces errors.

5 Case Study

To demonstrate the necessity of categorizing question-solving approaches, we present examples in Figure 4 and Figure 5 where SQL Developer (based on RAG) and SQL Researcher (based on question decomposition and CoT) are faced with simple and complex questions, respectively. In the simple question, the task is simply to find the maximum value of "$charge\_amount$". However, the method based on question decomposition and CoT breaks down the question, leading to incorrect identification of "$charge\_type$" in the second sub-question. In the complex question, the RAG-based method fails to understand the meaning of "$Bachelors\ program$" and consequently uses the non-existent column name "$Bachelors$". Each of these two approaches has its strengths for questions of different difficulty levels. It is necessary to categorize questions and address them with different approaches.

6 Conclusion

In this paper, we introduce a novel Text-to-SQL framework, called QCMA-SQL. It selects the appropriate agent to generate SQL based on the questions' varying difficulty levels. The proposed


Question: What are the first, middle, and last names for everybody enrolled in a Bachelors program?

Difficulty: Complex

Gold SQL:
```
SELECT DISTINCT T1.first_name , T1.middle_name , T1.last_name FROM Students AS T1 JOIN Student_Enrolment AS T2 ON
T1.student_id = T2.student_id JOIN Degree_Programs AS T3 ON T2.degree_program_id = T3.degree_program_id WHERE
T3.degree_summary_name = 'Bachelor'
```

SQL Developer(RAG):
```
SELECT students.first_name, students.middle_name, students.last_name FROM students JOIN student_enrolment ON
students.student_id = student_enrolment.student_id JOIN degree_programs ON student_enrolment.degree_program_id =
degree_programs.degree_program_id WHERE degree_programs.degree_summary_name = 'Bachelors'                     ✗
```

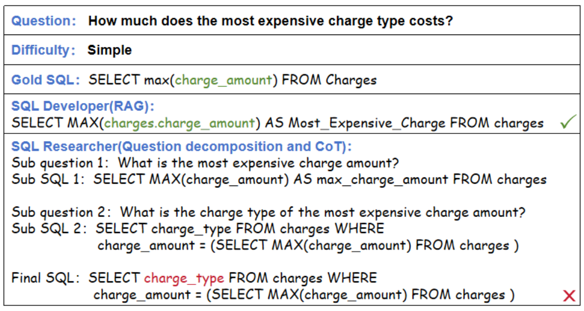
SQL Researcher(Question decomposition and CoT):
```
Sub question 1:  Identify the degree_program_id(s) for Bachelors programs.
Sub SQL 1:  SELECT degree_program_id FROM degree_programs WHERE degree_summary_name LIKE 'Bachelors%'

Sub question 2: Find the student_id(s) of students enrolled in Bachelors programs.
Sub SQL 2:  SELECT student_id FROM student_enrolment WHERE degree_program_id IN (SELECT degree_program_id FROM
degree_programs WHERE degree_summary_name LIKE 'Bachelor%' )

Sub question 3: Retrieve the first, middle, and last names of students enrolled in Bachelors programs.
Sub SQL 3:  SELECT first_name, middle_name, last_name FROM students WHERE student_id IN (SELECT student_id FROM
student_enrolment WHERE degree_program_id IN (SELECT degree_program_id FROM degree_programs WHERE
degree_summary_name LIKE 'Bachelor%'))

Final SQL: SELECT DISTINCT s.first_name, s.middle_name, s.last_name FROM students s JOIN student_enrolment se ON
s.student_id = se.student_id JOIN degree_programs dp ON se.degree_program_id = dp.degree_program_id WHERE
dp.degree_summary_name LIKE 'Bachelor%'                                                                      ✓
```

Figure 4: Complex question case.


Question: How much does the most expensive charge type costs?

Difficulty: Simple

Gold SQL: `SELECT max(charge_amount) FROM Charges`

SQL Developer(RAG):
```
SELECT MAX(charges.charge_amount) AS Most_Expensive_Charge FROM charges   ✓
```

SQL Researcher(Question decomposition and CoT):
```
Sub question 1:  What is the most expensive charge amount?
Sub SQL 1:  SELECT MAX(charge_amount) AS max_charge_amount FROM charges

Sub question 2:  What is the charge type of the most expensive charge amount?
Sub SQL 2:  SELECT charge_type FROM charges WHERE
            charge_amount = (SELECT MAX(charge_amount) FROM charges )

Final SQL:  SELECT charge_type FROM charges WHERE
            charge_amount = (SELECT MAX(charge_amount) FROM charges )         ✗
```

Figure 5: Simple question case.

framework not only develops an innovative question classification and schema linking model but also fully leverages the powerful semantic understanding capabilities of LLMs. Benefiting from the proposed schema selector, QCMA-SQL excels in handling questions of any difficulty level. Moreover, question classification has been proven to enhance the execution accuracy of the Text-to-SQL task. In summary, QCMA-SQL has achieved significant performance improvements in evaluation metrics and outperforms all comparative baselines.

7 Limitations

Due to the use of PLMs as encoders with limited context windows, limitations may be faced in generalizing to larger scale databases. In addition, there is room for improving the accuracy of vector database recall examples. Meanwhile, LLMs are sensitive to the way it is prompted and further exploration is needed to find the best strategy for different use cases.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shuaichen Chang and Eric Fosler-Lussier. 2023. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. In *NeurIPS 2023 Second Table Representation Learning Workshop*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Xiang Deng, Ahmed Hassan, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350.

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

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. 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 2505–2515.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931.

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. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang. 2023. Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain. In *International Conference on Neural Information Processing*, pages 341–356. Springer.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv preprint arXiv:2406.08426*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

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. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13076–13084.

R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, J Li, J Chim, et al. 2023c. Starcoder: May the source be with you! *Transactions on machine learning research*.

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*.

Qingkai Min, Yuefeng Shi, and Yue Zhang. 2019. A pilot study for chinese sql semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3652–3658.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

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.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

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

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901.

Ruoxi Sun, Sercan O Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. Sql-palm: Improved large language modeladaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-sql. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5376–5393.

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.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Lihan Wang, Bowen Qin, Binyuan Hui, Bowen Li, Min Yang, Bailin Wang, Binhua Li, Jian Sun, Fei Huang, Luo Si, et al. 2022. Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1889–1898.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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*. Association for Computational Linguistics.

A Experimental results from other LLMs

In addition to deepseek-v3 , we evaluated the QCMA-SQL framework on several LLMs. This helps to reveal the extent of performance degradation between LLMs with different capabilities. Due to hardware constraints, we were not able to deploy large open source LLM, so we called some common LLM APIs. The experimental results are shown in Table 7, which shows that the performance of the QCMA-SQL framework depends on the capability of LLM. Among the several LLMs on which the experiments were conducted, moonshot-v1-32k has the worst performance, and gpt-4-turbo has a slightly lower performance than deepseek-v3.

Model	EX
moonshot-v1-32k	80.6
gpt-3.5-turbo-0125	82.8
glm-4-air	83.5
glm-4-plus	84.2
glm-zero-preview	84.9
gpt-4-turbo	86.1
deepseek-v3	**87.4**

Table 7: Experimental results of QCMA-SQL framework based on different LLMs on Spider.

B Ablation study with modules in the schema selector

In QCMA-SQL, the schema selector plays a very important role. In order to study the principles of question classification and schema selection, our study evaluates the contribution of Bi-LSTM and multiple cross-attention in schema selector.

As shown in Table 8, among the four Bi-LSTMs, D-Bi-LSTM contributes the most to question classification, T-Bi-LSTM contributes the most to table classification, and C-Bi-LSTM contributes the most to column classification. Among the 3 cross-attention mechanisms, D_Q cross-attention has the greatest impact on question classification and T_C cross-attention mechanism has the greatest impact on schema selection.

Method	Table	Column	Difficulty
Schema Selector	**0.9984**	**0.9973**	**0.9893**
w/o D-Bi-LSTM	0.9938	0.9893	0.9674
w/o Q-Bi-LSTM	0.9906	0.9854	0.9722
w/o T-Bi-LSTM	0.9757	0.9796	0.9823
w/o C-Bi-LSTM	0.9851	0.9753	0.9785
w/o D_Q cross-attention	0.9860	0.9828	0.9616
w/o Q_T cross-attention	0.9784	0.9769	0.9734
w/o T_C cross-attention	0.9711	0.9725	0.9702

Table 8: Bi-LSTM and multiple cross-attention contributions to question classification and schema selection