EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

KADP-SQL: Knowledge-Augmented Generation and Dual-Path Validation in Text-to-SQL

Jun Peng[1], Jinguo You[1,*], Xiang Li[1], Jiaman Ding[1], Lianyin Jia[1]

[1]Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming, 650500, Yunnan, China

Abstract

The widespread application of Large Language Models (LLMs) has significantly advanced the development of the Text-to-SQL task, which aims to translate natural language questions posed by users into executable structured query statements. This allows non-expert users to efficiently access databases. However, existing models still struggle to generate correct SQL statements in the absence of sufficient knowledge, particularly under complex query scenarios, where they are prone to syntactic errors and semantic deviations. Moreover, most current approaches rely solely on execution feedback for SQL correction via a single path, making it difficult to detect semantic errors. To address these challenges, we propose KADP-SQL, which divides the Text-to-SQL task into two main modules: Knowledge-Augmented Generation and Dual-Path Validation. This framework structurally represents manually annotated evidence and incorporates web-based external knowledge to enhance SQL generation. Additionally, we introduce a dual-path SQL validation method to detect and correct errors in generated SQL queries. Through extensive experiments conducted on multiple closed-source and open-source LLMs, our proposed KADP-SQL achieves an execution accuracy of 70.53% on the BIRD development set and 88.22% on the Spider test set. These results demonstrate the effectiveness and adaptability of the proposed method.

Received on 06 October 2025; accepted on 11 January 2026; published on 02 February 2026

Keywords: LLM, Text-to-SQL, Knowledge-Augmented Generation, Dual-Path Validation

doi:10.4108/eetsis.10499

1. Introduction

The Text-to-SQL task aims to translate natural language questions posed by users into executable Structured Query Language (SQL) statements within a database, thereby enabling non-expert users to efficiently access and utilize structured data [1]. This task not only holds significant practical value but has also witnessed remarkable progress in recent years with the rise of large language models (LLMs) [2, 3].

Although LLMs have achieved significant success in the field of Text-to-SQL, it remains challenging to accurately convert user questions into SQL statements in the absence of specific domain knowledge, relevant rules, and explanatory external information. Even with state-of-the-art LLMs, generalization across different datasets still needs improvement [4]. As illustrated in Figure 1, when only the existing information is used as input to the LLMs, it may generate incorrect SQL. For example, the incorrect arithmetic operation "T1.'Free Meal Count (Ages 5-17)' / T1.'Enrollment (Ages 5-17)'" may lead to an empty result. In contrast, when external knowledge is provided, the LLMs is able to perform the correct computation and generate the correct SQL statement. To address errors in SQL generation caused by the lack of knowledge, some studies introduce formulaic knowledge [5], encoding domain knowledge into a structured form to build a retrievable knowledge base, which is then used to support SQL generation. While this approach can enhance model performance, it requires extensive manual annotation, resulting in very low efficiency.

Knowledge-to-SQL [6] proposes automatically generating "expert knowledge" for each query based on the user question and database schema to aid SQL generation. KAT-SQL [7] introduces a method for building a

*Corresponding author. Email: jgyou@kust.edu.cn

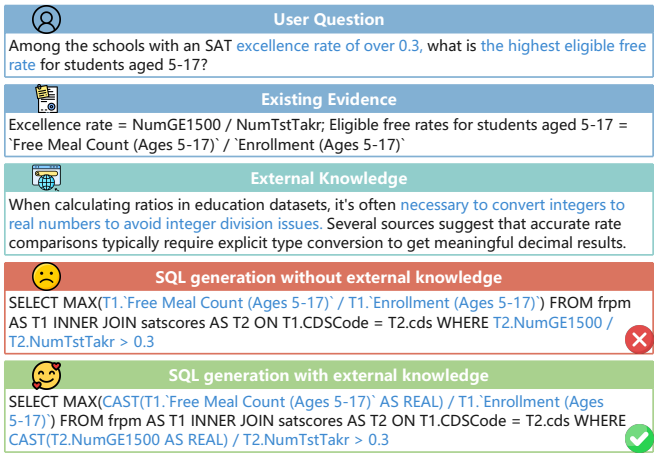

Figure 1. An example of SQL generation enhanced by external knowledge. Without external knowledge, the arithmetic operation was incorrectly handled. The external knowledge compensates for the LLM's lack of domain understanding, enabling it to generate the correct SQL.

knowledge base using LLMs, retrieving the most relevant entries as prompts during SQL generation. Although these methods extend the scope of knowledge available for SQL generation to some extent, their sources are relatively singular, relying entirely on the LLMs internal training data. Moreover, the cost of constructing such knowledge bases is high, the reusability of knowledge is limited, and the retrieval process requires the design of precise and efficient mechanisms. Additionally, due to the hallucination problem inherent in LLMs [8], the knowledge generated may appear plausible but actually be incorrect, and the generation process still depends on manually crafted few-shot prompts. In certain real-world applications, knowledge must also possess a degree of timeliness.

While the BIRD dataset has provided manually annotated evidence, existing studies have pointed out certain shortcomings in these annotations [9]. For example, the development set contains cases of missing or incorrectly labeled evidence. This research also confirms that omitting evidence leads to a significant drop in model performance. Moreover, error analyses in studies such as CHESS [10], MCS-SQL [11], and MAC-SQL [12] reveal that large language models sometimes fail to properly utilize the manually annotated evidence. On the other hand, earlier datasets like Spider [13] did not provide any form of evidence at all.

Current LLMs possess outstanding capabilities in language understanding and code generation. However, when faced with complex query intents, the SQL they generate still tends to suffer from syntax errors, semantic deviations, and execution failures, which severely limit the usability and robustness of these models in real-world scenarios [14]. To address this issue, researchers have proposed various post-processing strategies to improve the quality and execution accuracy of the final SQL output.

DIN-SQL [15] introduced a zero-shot self-correction module that applies two different prompting strategies to correct SQL without execution. MAC-SQL [12] designed a Refiner agent that inspects and corrects SQL based on execution feedback. Other frameworks such as OpenSearch-SQL [16], XiYan-SQL [17], CHASE-SQL [18], CSC-SQL [19], Fin-SQL [20], PET-SQL [21], C3 [22], DAIL-SQL [2], CHESS [10], and MCS-SQL [11] adopt another approach: generating multiple candidate SQL queries for a natural language question, using execution results to correct or eliminate candidates, and selecting the most consistent one through a consistency strategy or choosing one as the final SQL using a fine-tuned selection model. Although these methods improve model performance, they rely on a single correction path during SQL post-processing, making it difficult to detect semantic errors. Additionally, they depend on multiple candidate SQLs, which increases the cost and time required to generate the final SQL query and poses challenges for practical deployment.

To address these challenges, we propose a Text-to-SQL architecture named KADP-SQL. This architecture utilizes knowledge to enhance the preliminary SQL generation and adopts a dual-path verification strategy to check and correct the preliminary SQL, aiming to identify and fix as many erroneous SQLs as possible.

Specifically, to maximize the use of existing knowledge, we represent the existing evidence in the dataset in a structured form as SQL queries, so that the LLMs can better understand it. To avoid errors in SQL generation caused by lack of knowledge, we introduce external knowledge via web search, accessing high-quality web pages through search engines and retrieving highly relevant content. This also enables real-time information retrieval, improving the model's adaptability across different domains. During the preliminary SQL generation, we integrate both types of knowledge into the prompt to improve the quality and correctness of the preliminary SQL. To identify SQLs with syntax and semantic errors as much as possible, we introduce a dual-path verification strategy to check SQL correctness. In the first verification path, to identify the specific error locations as much as possible, we first decompose the natural language question into a query goal and sub-questions, then check whether the preliminary SQL satisfies each part and provide revision suggestions for the incorrect parts. In the second verification path, to identify errors in complex queries as much as possible, we convert the preliminary SQL into a natural language query plan, propagate and amplify potential errors in the SQL into the query plan, and then analyze the SQL's correctness based on the query plan and provide modification suggestions for the incorrect

parts. Finally, we correct the erroneous SQL using the results from the dual-path verification, and iteratively optimize the SQL through the checking and correction modules to obtain the final SQL.

We conducted extensive experiments of KADP-SQL on both the BIRD and Spider datasets, validating its effectiveness with both open-source and closed-source LLMs. The experimental results demonstrate the effectiveness of the proposed KADP-SQL. Our main contributions are summarized as follows:

(1) We propose a novel knowledge-augmented generation method that improves the quality and correctness of preliminary SQL generation in Text-to-SQL models, as well as their adaptability across different domains;

(2) This paper introduces a dual-path SQL accuracy validation method, which offers stronger generality and error detection capability compared to traditional single-path checking approaches;

(3) Our method, while generating only a single SQL query, achieves an execution accuracy of 70.53% on the development set of the BIRD dataset and 88.22% on the test set of the Spider dataset.

The remainder of this paper is structured as follows. Section 2 presents the existing methods relevant to this study. Section 3 elaborates on the design of the proposed model framework and its implementation details. Section 4 evaluates the performance of the proposed method through various experiments and analyzes the effectiveness of each module. Finally, Section 5 concludes the paper and points out the future work.

2. Related Work

2.1. Prompt Engineering in Text-to-SQL

Early studies have confirmed that prompt engineering can effectively optimize the performance of LLMs [23, 24]. Research methods such as Chain-of-Thought (CoT) [25, 26] and Retrieval-Augmented Generation (RAG) [27] have enhanced the reasoning capabilities of LLMs by providing enriched contextual information. The DAIL-SQL [2] system analyzed the impact of different problem formulations, example selections, and prompt structures on SQL generation performance, demonstrating that well-designed prompts can significantly improve both performance and efficiency. Meanwhile, several studies [3, 4, 28–30] have validated the effectiveness of CoT prompting in Text-to-SQL tasks. AP-SQL [31] incorporates Chain-of-Thought (CoT) and Graph-of-Thought (GoT) to enhance model performance in SQL generation. Works such as DIN-SQL [15], MAC-SQL [12], SQL-PaLM [32], and C3 [22] leverage zero-shot and few-shot prompts, using carefully designed templates to guide LLMs in generating high-quality SQL. Some studies [33] also employ least-to-most prompting to address complex problems.

Schema linking is also a widely adopted strategy, which reduces noise and computational overhead by selecting only the relevant schema elements as input. TA-SQL [34] and PET-SQL [21] first use an LLM to generate a preliminary SQL query and then extract the related tables and columns from it as the subsequent schema input. RSL-SQL [35] proposes a bidirectional schema linking strategy: in the forward direction, it uses an LLM to directly select the relevant tables and columns, while in the backward direction, it uses an LLM to generate a preliminary SQL query to extract the schema, thereby supplementing any key tables or columns that may have been missed in the forward linking. Finally, the results of the forward and backward linking are merged to obtain the final simplified schema. Solid-SQL [36] uses an LLM to create an augmented dataset for fine-tuning schema linking, and then performs schema prediction using the fine-tuned model. SchemaGraphSQL [37] and LGESQL [38] store schemas in a graph database and retrieve relevant sub-schemas from it. However, schema linking still carries substantial risks, because if any necessary elements are missed, the final result will also be wrong. In addition, relying solely on prompt engineering may be insufficient for handling queries with different levels of complexity, which makes it necessary to use external knowledge for enhancement, and the use of external knowledge can improve SQL generation even without performing schema linking.

2.2. Multi-Candidate Generation in Text-to-SQL

With the rapid development of LLMs, multi-candidate generation methods have become one of the mainstream approaches in the Text-to-SQL field. To obtain the correct SQL query as much as possible, these methods generate multiple candidate SQL statements using different prompt styles, different few-shot examples, and different models. Then, they select the final SQL statement from the candidate set using self-consistency, model-based ranking, or dedicated selector models. Representative methods include OpenSearch-SQL [16], XiYan-SQL [17], CHASE-SQL [18], CSC-SQL [19], CHESS [10], Agentar-Scale-SQL [39], and CM-SQL [40]. Because their candidate sets provide broad coverage, these approaches have achieved high performance on both the BIRD and Spider datasets.

However, this approach faces substantial computational resource and cost challenges. Generating a large number of candidate SQL statements requires significant computational resources, and if each candidate SQL must undergo correctness verification, the overall cost grows exponentially. In addition, many equivalent SQL statements may exist in the candidate set, resulting

in redundancy. The final selection also presents great challenges, making it difficult to choose the correct SQL.

2.3. Knowledge-Augmented Text-to-SQL

Recent studies have explored the use of knowledge to enhance Text-to-SQL models. For example, [5] constructed a formulaic knowledge bank and applied appropriate manual post-processing. Subsequently, [4] released the BIRD benchmark dataset, in which most questions are accompanied by manually annotated knowledge to support SQL generation. However, this type of annotation is costly and heavily dependent on domain experts. To overcome the bottleneck of manual annotation, [6] proposed the Knowledge-to-SQL framework, which fine-tunes a model to automatically generate expert knowledge that is then used to enhance SQL generation. SEED [9] analyzed the deficiencies of manually annotated evidence in the BIRD benchmark dataset and utilized few-shot prompting to guide LLMs in automatically generating evidence. Similarly, [7] utilized few-shot prompting to guide the LLM in constructing a reusable knowledge base, from which relevant content is retrieved during SQL generation for enhancement. [41] extracts schema knowledge from the database and manually converts it into training data usable by LLMs, and the constructed knowledge data is eventually used to pre-train the LLM to enhance SQL generation. SLENet [42] uses similarity retrieval to obtain knowledge that is semantically similar to the current question from public datasets such as WikiSQL [43] to enhance SQL generation. LPE-SQL [44] builds an expandable auxiliary knowledge base from the Text-to-SQL training set and then retrieves similar example knowledge from the knowledge base to enhance SQL generation. However, although these methods can improve model performance, they require manual annotation, which leads to low efficiency, and the cost of constructing a knowledge base is relatively high. In addition, they rely entirely on the internal corpus of LLMs, which often introduces certain limitations during knowledge generation.

2.4. SQL Check and Correction

In the field of Text-to-SQL, analyzing the correctness of SQL statements and correcting them is a critically important research direction. DIN-SQL [15] first proposed a zero-shot self-correction mechanism, which corrects SQL statements through carefully designed prompts without executing them. MAC-SQL [12] introduced a multi-agent collaboration strategy, where the Refiner agent diagnoses SQL statements by invoking tools to obtain execution feedback and combining it with self-consistency, and then corrects the faulty SQL accordingly. SEA-SQL [45] designed an execution feedback-driven iterative correction mechanism, which checks and corrects SQL statements based on execution results under a zero-shot setting. SQLFixAgent [46] incorporates an SQLRefiner agent to inspect and correct SQL statements; this agent calls tools to obtain database execution feedback to check the SQL, and if the check fails, it performs the correction. Similarly, [47] modifies incorrect SQL using database error messages and finally selects the best SQL statement through a voting mechanism across multiple models. Moreover, multi-candidate methods such as OpenSearch-SQL [16], XiYan-SQL [17], CHASE-SQL [18], CSC-SQL [19], and CHESS [10] also leverage execution results to correct candidate SQLs. However, existing studies often rely on the self-consistency strategy of LLMs to determine SQL correctness. During the correction process, they typically only rewrite and fix SQL statements that fail due to syntax errors, while lacking effective mechanisms to identify and correct queries that are executable but semantically incorrect.

3. Methods

3.1. Overall Framework

This section provides an overview of the proposed KADP-SQL framework, which consists of five main components: (1) Evidence-to-SQL conversion; (2) Web search for external knowledge; (3) Preliminary SQL generation; (4) SQL correctness checking; and (5) SQL correction. KADP-SQL first utilizes an LLM to convert the manually annotated evidence in the dataset into structured SQL queries. It then retrieves external knowledge related to the question through a web search API. The structured evidence and the retrieved external knowledge are jointly provided to the LLM to generate a preliminary SQL statement. This preliminary SQL is passed to the SQL checking module for dual-path validation. If any issues are identified, the SQL correction module is used to revise the SQL. Through iterative optimization between the checking and correction modules, the final SQL statement is produced. The overall architecture of KADP-SQL is illustrated in Figure 2, and the following sections delve into the details of each component.

3.2. Evidence-to-SQL

In this section, we first extract the manually annotated evidence from the dataset and provide the database schema together with the evidence as input to the LLMs, prompting the model to convert the evidence into a structured SQL query. The generated SQL is then executed on the corresponding database to obtain execution feedback. In cases where the result set is too long, only the first six rows are retained as the final execution feedback.

To minimize errors in the SQL generated from the structured representation of the evidence, we adopt

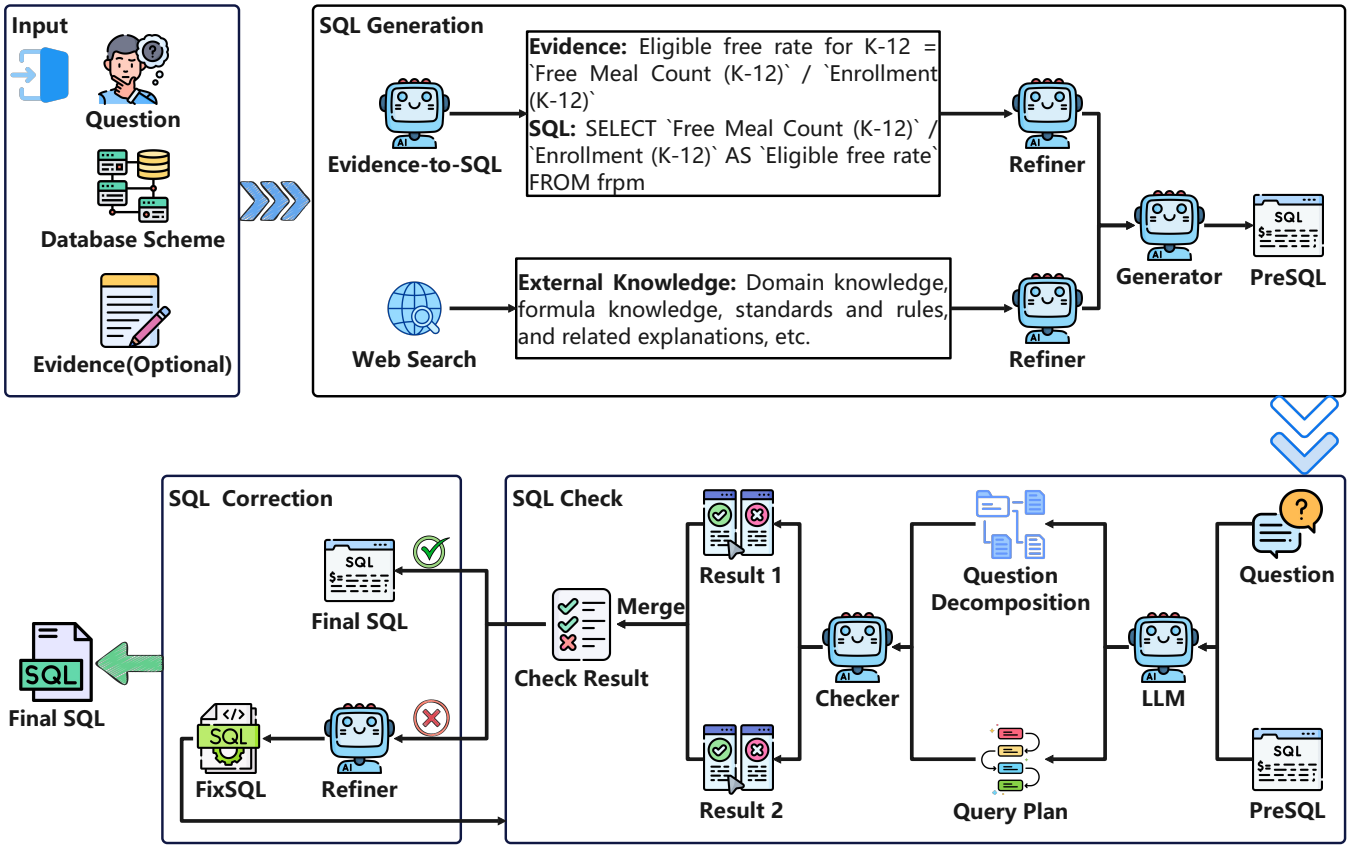

Figure 2. Overview of the KADP-SQL framework, which is composed of five parts: (1) Evidence-to-SQL, which formats the provided evidence into SQL statements; (2) Web Search, which retrieves relevant external knowledge through web search; (3) Preliminary SQL Generation (PreSQL), enhancing SQL generation using external knowledge; (4) SQL Correctness Checking, which verifies SQL correctness through dual-path validation; and (5) SQL Correction, which refines SQL queries that fail validation based on the checking results.

an execution-guided strategy to optimize the SQL statements. Specifically, we input both the SQL and its execution result into the LLM, prompting the model to analyze the correctness of the SQL and correct any errors to ensure that the evidence is accurately represented in SQL form. Since the evidence typically involves only a small number of tables and columns, the SQL is less prone to error, so only a single round of correction is performed. Manually annotated evidence often varies in format and style, and frequently references elements in the database schema. Converting it into structured SQL format helps the LLM better understand and utilize the information.

3.3. Web Search for External Knowledge

Although the BIRD dataset provides evidence for most questions, some questions still lack relevant knowledge or are annotated with incorrect information. Moreover, datasets like Spider do not provide any related knowledge at all, which may lead to various errors during SQL generation. Therefore, we propose retrieving external knowledge through web search, aiming to find relevant external information for each user's question.

In this component, we input the user question and database schema into a web search API to guide the retrieval of external knowledge required for SQL generation. To avoid redundancy, we specifically restrict the search to content that does not overlap with the existing knowledge. Specifically, the retrieved external knowledge includes domain knowledge, formulaic knowledge, standards and rules, and related explanatory information. For each question, we perform an independent search to retrieve its corresponding external knowledge, ensuring relevance and further reducing errors caused by knowledge gaps. In addition, to prevent the retrieved knowledge from being overly redundant, we utilize an LLM to further refine and summarize the search results, eliminating unnecessary information.

3.4. Preliminary SQL Generation

During the preliminary SQL generation process, in order to improve the quality and accuracy of the preliminary SQL, we make full use of the results obtained from the previous two components. We input the structured SQL generated from evidence in the first part, the external knowledge retrieved in the second part, the question, and the database schema into the LLMs, prompting it to generate the preliminary SQL statement. To enhance the LLMs' understanding of the evidence, we also include the original evidence in the prompt to guide the SQL generation.

3.5. SQL Check

To improve the accuracy of the generated preliminary SQL as much as possible, we propose a dual-path accuracy verification method to validate the correctness of the SQL. The divide-and-conquer strategy decomposes a complex problem into smaller subproblems, solves each subproblem individually, and then combines their solutions to obtain the final answer. Inspired by this idea, we adopt a reverse thinking approach in the first verification path by decomposing the original question for backward validation. Specifically, the question is provided as input to the LLM, and Few-shot learning is used to guide the LLM in performing the decomposition. The user question is broken down into a query target and sub-questions. Then, both the preliminary SQL and the decomposition result are fed into the LLM, prompting it to verify whether the SQL satisfies each component of the decomposition. During the verification process, we use Few-shot examples. To avoid the examples affecting the LLM's judgment and reasoning, we use three examples with a final verification result of YES and three examples with a result of NO as demonstrations.

The query plan (also known as the execution plan) is a detailed set of execution steps generated by a relational database optimizer that transforms an abstract query request into a series of concrete operations. This plan details the methods for accessing tables, the join strategies, and the specific operations performed on the data. CHASE-SQL [18] employs a chain-of-thought based on query plans to guide LLMs in generating candidate SQL queries. Inspired by this approach, in the second verification path we use the query plan to check the correctness of the SQL. We first input the preliminary SQL into the LLM and then use a pre-designed Few-shot prompt to guide the LLM to convert the SQL statement into a query plan described in natural language. In this step, we instruct the LLM to represent all the syntax and semantic information of the preliminary SQL in the query plan so that any potential errors in the SQL are fully reflected. Next, the preliminary SQL and its query plan are fed into the LLM, and similar to the first path, we use three sample cases with a final check result of YES and three sample cases with a final check result of NO to guide the LLM in judging and analyzing the correctness of the SQL. The query plan magnifies potential errors in the SQL and clearly describes all the steps of the SQL, which is beneficial for the LLM to judge and analyze the correctness of the SQL.

In all verification paths, to accurately detect syntax errors or empty result sets, we adopt an execution-guided strategy to check the SQL. The SQL statement is executed on the corresponding database to obtain execution feedback. In cases where the result set is too long, only the first six rows are retained as the final execution feedback. To identify as many erroneous SQL statements as possible, we merge the verification results from both paths: if either path determines that the SQL is incorrect, the SQL is classified as erroneous. To enable the correction module to better revise the SQL statements, for each SQL identified as problematic, we prompt the LLM to first provide its judgment and then offer modification suggestions for the SQL.

3.6. SQL Correction

In this module, we correct the SQL statements identified as erroneous during the verification process. First, the verification results from the checking module are used as error analysis, and then combined with the preliminary SQL into the prompt to guide the LLM in performing the correction. To prevent the model from repeating the same mistakes across multiple correction attempts, we maintain a history of previous modifications and include this history in the next prompt to guide the model during further corrections. Through iterative refinement between the checking and correction modules, the final SQL statement is obtained. We set the maximum number of iterations to three.

4. Experiments

4.1. Datasets

We evaluate the effectiveness of the proposed KADP-SQL on two widely used Text-to-SQL benchmark datasets: Spider and BIRD.

The Spider dataset is a large-scale, cross-domain Text-to-SQL benchmark constructed through manual annotation. It contains 10,181 natural language questions paired with 5,693 complex SQL queries. Spider spans 200 different databases across 138 distinct domains, with each database comprising multiple tables and rich inter-table relationships. The training and testing sets use entirely different database schemas and query structures, making it a rigorous benchmark for evaluating a model's cross-domain generalization ability.

The BIRD dataset is a large-scale, cross-domain Text-to-SQL benchmark designed for real-world application

Table 1. Execution accuracy of KADP-SQL on the BIRD development set across different models and difficulty levels.

Model	Simple (925)	Moderate (464)	Challenging (145)	Total (1534)
Qwen-2.5-coder-32b	65.41	49.35	43.45	58.47
KADP-SQL + Qwen-2.5-coder-32b (ours)	**71.89**	**58.84**	**57.24**	**66.56** (↑ **8.09**)
GPT-4.1-mini	68.43	53.02	49.66	61.99
KADP-SQL + GPT-4.1-mini (ours)	**72.65**	**58.84**	**62.76**	**67.54** (↑ **5.55**)
Grok-3-mini	67.14	54.74	48.97	61.67
KADP-SQL + Grok-3-mini (ours)	**73.51**	**60.99**	**56.55**	**68.12** (↑ **6.45**)
DeepSeek-V3	68.86	54.96	47.59	62.65
KADP-SQL + DeepSeek-V3 (ours)	**73.84**	**62.07**	**55.86**	**68.58** (↑ **5.93**)
Gemini-2.0-flash	71.46	53.88	53.79	64.47
KADP-SQL + Gemini-2.0-flash (ours)	**75.57**	**62.50**	**64.14**	**70.53** (↑ **6.06**)

Table 2. Execution accuracy of KADP-SQL on the Spider test set across different models and difficulty levels.

Model	Count	Easy (470)	Medium (857)	Hard (463)	Extra (357)	All (2147)
Qwen-2.5-coder-32b		88.51	85.76	76.89	68.91	81.65
KADP-SQL + Qwen-2.5-coder-32b (ours)		**93.40**	**90.08**	**82.94**	**78.15**	**87.28** (↑ **5.63**)
Gemini-2.0-flash		91.91	88.10	80.56	74.51	85.05
KADP-SQL + Gemini-2.0-flash (ours)		**93.19**	**89.96**	**86.83**	**79.27**	**88.22** (↑ **3.17**)

Table 3. Performance comparison of different methods on the BIRD development set.

Method	EX (%)
CHASE-SQL + Gemini-1.5-pro [18]	74.90
XiYan-SQL [17]	73.34
OpenSearch-SQL + GPT-4o [16]	69.30
OmniSQL-32B [48]	69.23
CHESS [10]	68.31
RSL-SQL + GPT-4o [35]	67.21
Distillery + GPT-4o [49]	67.21
E-SQL + GPT-4o [50]	65.58
MCS-SQL + GPT-4 [11]	63.36
KADP-SQL + Gemini-2.0-flash (ours)	**70.53**

Table 4. Performance comparison of different methods on the Spider test set.

Method	EX (%)
XiYan-SQL [17]	89.65
MCS-SQL + GPT-4 [11]	89.60
RSL-SQL + GPT-4o [35]	87.90
CHASE-SQL + Gemini-1.5 [18]	87.60
OmniSQL-32B [48]	87.60
CHESS [10]	87.20
OpenSearch-SQL + GPT-4o [16]	87.10
DAIL-SQL + GPT-4 [2]	86.60
DIN-SQL + GPT-4 [15]	85.30
KADP-SQL + Gemini-2.0-flash (ours)	**88.22**

scenarios. It contains 12,751 question-SQL pairs across 95 databases, with a total data volume of 33.4GB, covering 37 specialized domains such as finance, healthcare, sports, education, and blockchain. Unlike traditional evaluation benchmarks, BIRD introduces manually annotated external knowledge to enhance model understanding of questions, database schemas, and values, and it also provides database description files. Compared with Spider, BIRD includes fewer database types, but its SQL queries are more complex, requiring more advanced SQL operations and external knowledge to generate accurate SQL statements.

4.2. Evaluation Metrics

According to the evaluation criteria of the BIRD and Spider test suites, we use execution accuracy (EX) as the primary metric to evaluate the effectiveness of the framework. Execution accuracy is defined as the proportion of predicted SQL queries whose execution results exactly match those of the gold SQL queries, i.e., the percentage of queries that produce the same execution results as the gold SQL out of the total number of queries.

Table 5. Effectiveness of Evidence-to-SQL on the BIRD development set across different models and difficulty levels.

Model		Simple	Moderate	Challenging	Total
Qwen-2.5-coder-32b	+Evidence	65.41	49.35	43.45	58.47
	+Evidence-to-SQL	66.70 (+1.29)	52.80 (+3.45)	49.66 (+6.21)	**60.89 (+2.42)**
GPT-4.1-mini	+Evidence	68.43	53.02	49.66	61.99
	+Evidence-to-SQL	69.19 (+0.76)	55.39 (+2.37)	54.48 (+4.82)	**63.62 (+1.63)**
Grok-3-mini	+Evidence	67.14	54.74	48.97	61.67
	+Evidence-to-SQL	68.76 (+1.62)	55.82 (+1.08)	50.34 (+1.37)	**63.10 (+1.43)**
DeepSeek-V3	+Evidence	68.86	54.96	47.59	62.65
	+Evidence-to-SQL	69.84 (+0.98)	55.39 (+0.43)	49.66 (+2.07)	**63.56 (+0.91)**
Gemini-2.0-flash	+Evidence	71.46	53.88	53.79	64.47
	+Evidence-to-SQL	72.76 (+1.30)	54.31 (+0.43)	54.48 (+0.69)	**65.45 (+0.98)**

4.3. Baseline Model

In this paper, to comprehensively evaluate the performance of KADP-SQL, we conducted extensive experiments on multiple closed-source and open-source models. Specifically, experiments were carried out on Gemini-2.0-flash, DeepSeek-V3, Grok-3-mini, GPT-4.1-mini, and Qwen-2.5-coder-32b to verify the generalization ability and effectiveness of our proposed solution.

4.4. Results

Bird Results. We conducted experiments using several different baseline models on the development set of the BIRD dataset. As shown in Table 1, the results present the performance of the proposed KADP-SQL framework. When using the open-source model Qwen-2.5-coder-32b, our method achieved an overall execution accuracy improvement of 8.09% compared to the baseline model, with increases of 9.49% and 13.79% on the medium and challenging difficulty levels, respectively. This shows that our method performs well on open-source models.

When using closed-source LLMs, our method achieved higher performance. With Gemini-2.0-flash, the execution accuracy on the BIRD development set reached 70.53%. Specifically, compared to the baseline models, the execution accuracy improved by 5.55% on GPT-4.1-mini, 6.45% on Grok-3-mini, and by 5.93% and 6.06% on DeepSeek-V3 and Gemini-2.0-flash, respectively. These results indicate that KADP-SQL has generalization ability across different LLMs.

In addition, Table 3 presents a performance comparison between the KADP-SQL framework and other competing methods on the BIRD dataset. It can be observed that our method outperforms most existing approaches that rely on high-cost baseline models, even when using low-cost baseline models. Compared with methods that improve accuracy by generating multiple candidate SQL queries (such as OpenSearch-SQL, XiYan-SQL, and CHASE-SQL), our method achieves comparable execution accuracy while generating only a single SQL query, indicating that it remains competitive under a single-generation strategy.

Spider Results. To demonstrate the effectiveness of the proposed KADP-SQL on datasets without provided evidence, we also evaluated its performance on the test set of the Spider dataset. We used the default configuration of our method, with the only adjustment being the removal of the structured evidence representation component, as the Spider dataset does not include any evidence. As shown in Table 2, we conducted evaluations on both an open-source baseline model and a closed-source baseline model. When using the open-source model Qwen-2.5-coder-32b, our method achieved an execution accuracy of 87.28%, representing a 5.63% improvement over the baseline. With the closed-source model Gemini-2.0-flash, the execution accuracy reached 88.22%, an improvement of 3.17% over the baseline.

Furthermore, as shown in Table 4, we present a performance comparison between our method and other strong methods on the Spider dataset. Similar to the results on the BIRD dataset, our method outperforms most approaches based on high-cost models and achieves performance close to the leading methods. This highlights the generalizability of our approach across different datasets.

4.5. Effectiveness of Evidence-to-SQL

To verify the effectiveness of formatting manually annotated evidence into SQL statements, we conducted experiments on the BIRD dataset using multiple models. As shown in Table 5, for both open-source and closed-source models, converting evidence into SQL statements leads to performance improvements across all difficulty levels. These results confirm that transforming manually annotated evidence into SQL

Table 6. Effectiveness of Web Search on the Spider test set across different models and difficulty levels, and performance comparison with SEED.

Model		Easy	Medium	Hard	Extra	All
Qwen-2.5-coder-32b	No Knowledge	88.51	85.76	76.89	68.91	81.65
	SEED [9]	90.85 (+2.34)	87.51 (+1.75)	78.19 (+1.30)	**72.83 (+3.92)**	83.79 (+2.14)
	Web Search	**92.34 (+3.83)**	**88.21 (+2.45)**	**78.40 (+1.51)**	72.27 (+3.36)	**84.35 (+2.70)**
Gemini-2.0-flash	No Knowledge	91.91	88.10	80.56	74.51	85.05
	SEED [9]	91.28 (–0.63)	88.10	80.99 (+0.43)	**78.99 (+4.48)**	85.75 (+0.70)
	Web Search	**92.34 (+0.43)**	**88.80 (+0.70)**	**83.80 (+3.24)**	77.87 (+3.36)	**86.68 (+1.63)**

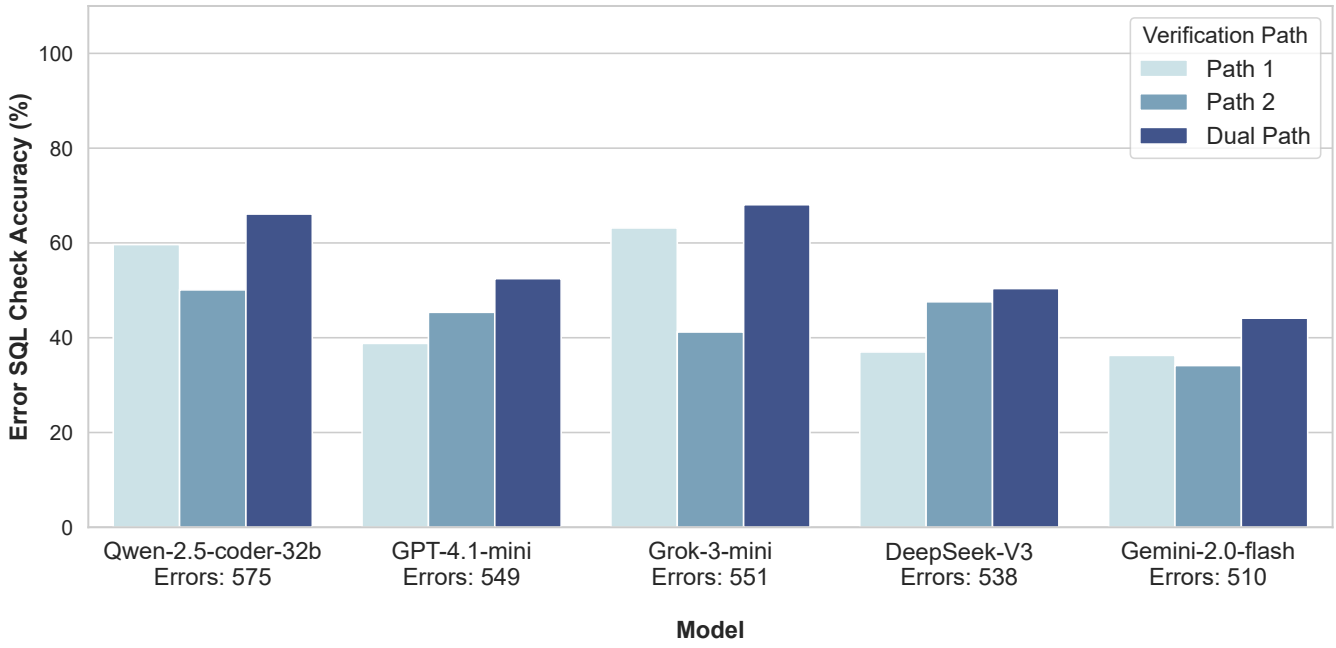

Figure 3. Effectiveness of dual-path SQL correctness checking on the BIRD development set across different models.

Table 7. Ablation results of KADP-SQL with different components on the BIRD development set.

Method	Execution Accuracy (%)	Δ(%)
KADP-SQL + Qwen-2.5-coder-32b	66.56	-
w/o Evidence-to-SQL	64.44	2.12
w/o Web Search	63.65	2.91
w/o Check & Correction	62.78	3.78
KADP-SQL + Gemini-2.0-flash	70.53	-
w/o Evidence-to-SQL	68.90	1.63
w/o Web Search	68.16	2.37
w/o Check & Correction	67.29	3.24

form can aid model understanding and enhance the generation of preliminary SQL queries.

4.6. Effectiveness of Web Search

In this section, we evaluate the effectiveness of web search for external knowledge on the test set of the Spider dataset and compare it with related methods. As shown in Table 6, we assess the performance of two models under three different settings: No Knowledge, with evidence generated by SEED, and with external knowledge obtained through web search. The results show that for both open-source and closed-source

models, although the evidence generated by SEED performs better than our method at the most challenging difficulty level, our method achieves greater overall improvement, demonstrating the effectiveness of the external knowledge we obtain.

4.7. Effectiveness of SQL Check

To verify the effectiveness of dual-path error checking, we conducted experiments on the BIRD dataset. As shown in Figure 3, we performed error checking on preliminary SQL generated with knowledge augmentation across multiple models and compared the accuracy of error detection under three settings: (1) Path 1: checking SQL using question decomposition; (2) Path 2: checking SQL using the query plan; (3) Dual Path: dual-path checking, where the SQL is judged as incorrect if either path identifies it as erroneous. The results show that Dual Path effectively combines the strengths of Path 1 and Path 2, demonstrating good performance in SQL correctness checking.

4.8. Ablation Study

To evaluate the impact of each component in the proposed KADP-SQL architecture on overall performance, we conducted ablation studies on the development set of the BIRD dataset using one open-source model and one closed-source model. As shown in Table 7, for the open-source model Qwen-2.5-coder-32b, removing the Evidence-to-SQL module resulted in a 2.12% drop in overall execution accuracy, removing the web search module led to a 2.91% drop, and removing the SQL checking and correction module caused a 3.78% decrease. It can be seen that excluding any module leads to performance degradation on the open-source model. Similarly, for the closed-source model Gemini-2.0-flash, the removal of the Evidence-to-SQL module, web search module, and SQL checking and correction module resulted in performance drops of 1.63%, 2.37%, and 3.24%, respectively, further confirming the importance of these components within the overall framework.

5. Conclusion and Future Work

In this paper, we propose a Text-to-SQL framework named KADP-SQL, which enhances Text-to-SQL performance through knowledge-augmented generation and dual-path validation. To improve the quality of preliminary SQL, we format manually annotated evidence into SQL representations to help the LLM better understand the evidence, and supplement the LLM's knowledge with external information retrieved via web search. These two sources of information are combined to enhance the generation of preliminary SQL. Additionally, to identify errors in SQL statements, we introduce a dual-path validation approach that verifies SQL correctness through question decomposition and query plan analysis. Through extensive experiments on two different types of datasets, we demonstrate the effectiveness of the proposed KADP-SQL.

Although KADP-SQL has shown promising effectiveness, there are still areas worth further exploration. In future work, mechanisms for verifying the validity of knowledge can be investigated to reduce the influence of invalid or redundant information. Moreover, when handling complex queries, the SQL checking module may still have limitations. Future research could consider incorporating multi-agent collaboration mechanisms or more fine-grained error detection methods to further enhance the model's ability to process complex queries.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this article.

Data Availability

The dataset used in this paper is available for download at the following link: BIRD: https://bird-bench.github.io/, Spider: https://yale-lily.github.io/spider.

Appendix A. Prompt Template

A.1. Prompt Template for Preliminary SQL Generation

```
### You are an experienced database expert. You are provided with a database schema and a natural language question below. Your task is to understand the schema and the question and generate a valid SQLite query to answer the question using the provided hints and external knowledge.

Database Schema:
{DATABASE_SCHEMA}

Question:
{QUESTION}

Hint:
{HINT}

Hint expressed in SQL statement:
{HINT_SQL}

External Knowledge:
{Web_Search_External_Knowledge}

### Instructions:
-- Ensure that only the columns needed to answer the question are selected, and no irrelevant columns are included.
-- Ensure that the final query returns all the information asked in the question.
-- Carefully consider whether the question requires unique values, and use 'DISTINCT' when necessary.
-- Only one SQLite query statement can be generated, not insert, delete, update, or create statements.

### Take a deep breath and think step by step to find the correct SQLite SQL query. If you follow all the instructions and generate the correct query, I will give you 1 million dollars.

### Please output in the following JSON format(You must ensure that the output result can be parsed by python's json.loads.):
{{
    "Reason": "The reason why you arrived at the final SQL query.",
    "SQL": "The SQL query you ultimately generated."
}}
```

A.2. Prompt Template for SQL Correctness Check in Path 1

```
### You are an experienced SQL database expert. Your task is to judge whether [Pre_SQL] satisfies each part of the question decomposition based on all the information provided.

[Database Schema]: {DATABASE_SCHEMA}

[Question]: {QUESTION}

[Hint]: {HINT}

[Hint expressed in SQL statement]: {HINT_SQL}

[External Knowledge]: {Web_Search_External_Knowledge}

[Pre_SQL]: {SQL}

[Execution result of Pre_SQL](When the execution result has multiple rows, take the first six rows): {EXECUTE_RESULT}

[Question Decomposition]: {QUESTION_DIVIDE_RESULT}

[Examples]: {FEW_SHOT}
```

Instructions:
-- Please refer to the example and, based on the execution results of the [Pre_SQL], the database schema, the hint, and external knowledge, determine whether the [Pre_SQL] satisfies each part of the question decomposition.
-- If the result is 'NO', a modification suggestion needs to be given in the reason.
-- You only need to give modification suggestions, but you do not need to give modified SQL.
-- The restatement should be consistent with the [Question Decomposition], and no modifications should be made.

Take a deep breath, think step by step, and make the right judgment. If you follow all the instructions and make the right judgment and analysis, I will give you $1 million.

Please output as follows(Just output content in the following format without any unnecessary explanation):
1. Targets: <Restate the targets in [Question Decomposition]>: <Determine whether Pre_SQL satisfies this part, and the result is 'YES' or 'NO'>;
   Reason: <Give the reason for judgment based on all the information provided, and give the modification opinion of Pre_SQL if it is not satisfied.>
2. Subquery_1: <Restate the subquery in [Question Decomposition] in sequence>: <Determine whether Pre_SQL satisfies this part, and the result is 'YES' or 'NO'>;
   Reason: <Give the reason for judgment based on all the information provided, and give the modification opinion of Pre_SQL if it is not satisfied.>
3. Subquery_2: <If there are multiple subqueries in the [Question Decomposition], please give them in order in this format. If there are no further subqueries, there is no need to reply to this section.>: <Determine whether Pre_SQL satisfies this part, and the result is 'YES' or 'NO'>;
   Reason: <Give the reason for judgment based on all the information provided, and give the modification opinion of Pre_SQL if it is not satisfied.>

Does [Pre_SQL] satisfy all parts of the question: <The result is 'YES' or 'NO'>;
Reason: <Give the reason for judgment based on all the information provided.>

A.3. Prompt Template for SQL Correctness Check in Path 2

You are an experienced SQL database expert. Your task is to determine whether the predicted SQL can correctly answer the question, given the database schema, a question, a predicted SQL query, query plan of predicted SQL, and some additional information.

Important Notes:
-- The Query Plan of Pre_SQL is not obtained from actual SQL execution. It was generated directly from the Pre_SQL text using an LLM. Therefore, the plan may contain or reflect semantic or structural errors present in [Pre_SQL] itself.
-- Use the Query Plan is only to help understand how [Pre_SQL] operates, so that you can make the right judgment.

[Database Schema]: {DATABASE_SCHEMA}

[Question]: {QUESTION}

[Hint]: {HINT}

[Hint expressed in SQL statement]: {HINT_SQL}

[External Knowledge]: {Web_Search_External_Knowledge}

[Pre_SQL]: {SQL}

[Execution result of Pre_SQL](When the execution result has multiple rows, take the first six rows): {EXECUTE_RESULT}

[Query plan of Pre_SQL]: {QUERY_PLAN}

[Examples]: {FEW_SHOT}

Instructions:

-- Please refer to the example, first analyze each step of the query plan of Pre_SQL, then combine the execution results of Pre_SQL, the database schema, the hint, and external knowledge to determine whether [Pre_SQL] can return the correct result for the question.
-- If the result is 'NO', a modification suggestion needs to be given in the reason.
-- You only need to give modification suggestions, but you do not need to give modified SQL.

Take a deep breath, think step by step, and make the right judgment. If you follow all the instructions and make the right judgment and analysis, I will give you $1 million.

Please output as follows(Just output content in the following format without any unnecessary explanation):
Does [Pre_SQL] return the correct result for the question: <Determine whether [Pre_SQL] can return the correct result for the question, and the result is 'YES' or 'NO'>;
Reason: <Give the reasons for the judgment. If the result is NO, please provide modification suggestions for [Pre_SQL].>

A.4. Prompt Template for SQL Correction

You are an experienced SQL database expert. Your task is to correct the predicted SQL given the database schema, a problem, the error analysis of the predicted SQL, and some additional information.

[Database Schema]:
{DATABASE_SCHEMA}

[Question]:
{QUESTION}

[Hint]:
{HINT}

[Hint expressed in SQL statement]:
{HINT_SQL}

[External Knowledge]:
{Web_Search_External_Knowledge}

[Pre_SQL]:
{SQL}

[Error analysis of Pre_SQL]:
{ERROR_ANALYSIS}

[SQL Correction History]:
{CORRECTION_HISTORY}

Take a deep breath, think step by step, and correct the SQL query. If you follow all the instructions and fix the query correctly, I will give you 1 million dollars.

Please output in the following JSON format(You must ensure that the output result can be parsed by python's json.loads.):
{{
    "Reason": "The reason why you arrived at the final SQL query.",
    "Corrected_SQL": "Your corrected SQL query."
}}

References

[1] Cai, R., Xu, B., Zhang, Z., Yang, X., Li, Z. and Liang, Z. (2018) An encoder-decoder framework translating natural language to database queries. In Lang, J. [ed.] *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (Stockholm, Sweden: AAAI Press): 3977–3983. doi:10.24963/IJCAI.2018/553.

[2] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B. and Zhou, J. (2024) Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.* **17**(5): 1132–1145. doi:10.14778/3641204.3641221.

[3] Rajkumar, N., Li, R. and Bahdanau, D. (2022) Evaluating the text-to-sql capabilities of large language models. *CoRR* **abs/2204.00498**. doi:10.48550/ARXIV.2204.00498.

[4] Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B. *et al.* (2023) Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23 (Red Hook, NY, USA: Curran Associates Inc.).

[5] Dou, L., Gao, Y., Liu, X., Pan, M., Wang, D., Che, W., Zhan, D. *et al.* (2022) Towards knowledge-intensive text-to-SQL semantic parsing with formulaic knowledge. In Goldberg, Y., Kozareva, Z. and Zhang, Y. [eds.] *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (Abu Dhabi, United Arab Emirates: Association for Computational Linguistics): 5240–5253. doi:10.18653/v1/2022.emnlp-main.350.

[6] Hong, Z., Yuan, Z., Chen, H., Zhang, Q., Huang, F. and Huang, X. (2024) Knowledge-to-SQL: Enhancing SQL generation with data expert LLM. In Ku, L.W., Martins, A. and Srikumar, V. [eds.] *Findings of the Association for Computational Linguistics: ACL 2024* (Bangkok, Thailand: Association for Computational Linguistics): 10997–11008. doi:10.18653/v1/2024.findings-acl.653.

[7] Baek, J., Samulowitz, H., Hassanzadeh, O., Subramanian, D., Shirai, S., Gliozzo, A. and Bhattacharjya, D. (2025) Knowledge base construction for knowledge-augmented text-to-SQL. In Che, W., Nabende, J., Shutova, E. and Pilehvar, M.T. [eds.] *Findings of the Association for Computational Linguistics: ACL 2025* (Vienna, Austria: Association for Computational Linguistics): 26569–26583.

[8] Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q. *et al.* (2025) A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.* **43**(2): 42:1–42:55. doi:10.1145/3703155.

[9] Yun, J. and Lee, S. (2025) SEED: enhancing text-to-sql performance and practical usability through automatic evidence generation. *CoRR* **abs/2506.07423**. doi:10.48550/ARXIV.2506.07423.

[10] Talaei, S., Pourreza, M., Chang, Y., Mirhoseini, A. and Saberi, A. (2024) CHESS: contextual harnessing for efficient SQL synthesis. *CoRR* **abs/2405.16755**. doi:10.48550/ARXIV.2405.16755.

[11] Lee, D., Park, C., Kim, J. and Park, H. (2025) MCS-SQL: Leveraging multiple prompts and multiple-choice selection for text-to-SQL generation. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B.D. and Schockaert, S. [eds.] *Proceedings of the 31st International Conference on Computational Linguistics* (Abu Dhabi, UAE: Association for Computational Linguistics): 337–353.

[12] Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z. *et al.* (2025) MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B.D. and Schockaert, S. [eds.] *Proceedings of the 31st International Conference on Computational Linguistics* (Abu Dhabi, UAE: Association for Computational Linguistics): 540–557.

[13] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J. *et al.* (2018) Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Riloff, E., Chiang, D., Hockenmaier, J. and Tsujii, J. [eds.] *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (Brussels, Belgium: Association for Computational Linguistics): 3911–3921. doi:10.18653/v1/D18-1425.

[14] Floratou, A., Psallidas, F., Zhao, F., Deep, S., Hagleither, G., Tan, W., Cahoon, J. *et al.* (2024) NL2SQL is a solved problem... not! In *14th Conference on Innovative Data Systems Research, CIDR 2024, January 14-17, 2024* (Chaminade, HI, USA: www.cidrdb.org).

[15] Pourreza, M. and Rafiei, D. (2023) DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M. and Levine, S. [eds.] *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

[16] Xie, X., Xu, G., Zhao, L. and Guo, R. (2025) Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. *Proc. ACM Manag. Data* **3**(3): 194:1–194:24. doi:10.1145/3725331.

[17] Gao, Y., Liu, Y., Li, X., Shi, X., Zhu, Y., Wang, Y., Li, S. *et al.* (2024) Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *CoRR* **abs/2411.08599**. doi:10.48550/ARXIV.2411.08599.

[18] Pourreza, M., Li, H., Sun, R., Chung, Y., Talaei, S., Kakkar, G.T., Gan, Y. *et al.* (2025) CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-sql. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, April 24-28, 2025* (Singapore: OpenReview.net).

[19] Sheng, L. and Xu, S. (2025) CSC-SQL: corrective self-consistency in text-to-sql via reinforcement learning. *CoRR* **abs/2505.13271**. doi:10.48550/ARXIV.2505.13271.

[20] Zhang, C., Mao, Y., Fan, Y., Mi, Y., Gao, Y., Chen, L., Lou, D. *et al.* (2024) Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. In Barceló, P., Sánchez-Pi, N., Meliou, A. and Sudarshan, S. [eds.] *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, June 9-15, 2024* (Santiago, Chile: ACM): 93–105. doi:10.1145/3626246.3653375.

[21] Li, Z., Wang, X., Zhao, J., Yang, S., Du, G., Hu, X., Zhang, B. *et al.* (2024) PET-SQL: A prompt-enhanced two-stage text-to-sql framework with cross-consistency.

CoRR **abs/2403.09732**. doi:10.48550/ARXIV.2403.09732.

[22] Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Chen, L., Lin, J. *et al.* (2023) C3: zero-shot text-to-sql with chatgpt. *CoRR* **abs/2307.07306**. doi:10.48550/ARXIV.2307.07306.

[23] Liu, A., Hu, X., Wen, L. and Yu, P.S. (2023) A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *CoRR* **abs/2303.13547**. doi:10.48550/ARXIV.2303.13547.

[24] Wei, J., Bosma, M., Zhao, V.Y., Guu, K., Yu, A.W., Lester, B., Du, N. *et al.* (2022) Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022* (Virtual Event: OpenReview.net).

[25] Liu, X. and Tan, Z. (2023) Divide and prompt: Chain of thought prompting for text-to-sql. *CoRR* **abs/2304.11556**. doi:10.48550/ARXIV.2304.11556.

[26] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H. *et al.* (2022) Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K. and Oh, A. [eds.] *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

[27] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H. *et al.* (2020) Retrieval-augmented generation for knowledge-intensive NLP tasks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. and Lin, H. [eds.] *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[28] Tai, C.Y., Chen, Z., Zhang, T., Deng, X. and Sun, H. (2023) Exploring chain of thought style prompting for text-to-SQL. In Bouamor, H., Pino, J. and Bali, K. [eds.] *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (Singapore: Association for Computational Linguistics): 5376–5393. doi:10.18653/v1/2023.emnlp-main.327.

[29] Chang, S. and Fosler-Lussier, E. (2023) How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. *CoRR* **abs/2305.11853**. doi:10.48550/ARXIV.2305.11853.

[30] Zhang, H., Cao, R., Chen, L., Xu, H. and Yu, K. (2023) ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. In Bouamor, H., Pino, J. and Bali, K. [eds.] *Findings of the Association for Computational Linguistics: EMNLP 2023* (Singapore: Association for Computational Linguistics): 3501–3532. doi:10.18653/v1/2023.findings-emnlp.227.

[31] Tang, Z., Ma, Q. and Wu, D. (2025) Auto prompt sql: a resource-efficient architecture for text-to-sql translation in constrained environments. *CoRR* **abs/2506.03598**. doi:10.48550/ARXIV.2506.03598.

[32] Sun, R., Arik, S.Ö., Muzio, A., Miculicich, L., Gundabathula, S.K., Yin, P., Dai, H. *et al.* (2024) Sql-palm: Improved large language model adaptation for text-to-sql. *Trans. Mach. Learn. Res.* **2024**.

[33] Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D. *et al.* (2023) Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, May 1-5, 2023* (Kigali, Rwanda: OpenReview.net).

[34] Qu, G., Li, J., Li, B., Qin, B., Huo, N., Ma, C. and Cheng, R. (2024) Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-sql generation. In Ku, L., Martins, A. and Srikumar, V. [eds.] *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024* (Association for Computational Linguistics): 5456–5471. doi:10.18653/V1/2024.FINDINGS-ACL.324, URL https://doi.org/10.18653/v1/2024.findings-acl.324.

[35] Cao, Z., Zheng, Y., Fan, Z., Zhang, X., Chen, W. and Bai, X. (2024) RSL-SQL: robust schema linking in text-to-sql generation. *CoRR* **abs/2411.00073**. doi:10.48550/ARXIV.2411.00073.

[36] Liu, G., Tan, Y., Zhong, R., Xie, Y., Zhao, L., Wang, Q., Hu, B. *et al.* (2025) Solid-sql: Enhanced schema-linking based in-context learning for robust text-to-sql. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B.D. and Schockaert, S. [eds.] *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025* (Association for Computational Linguistics): 9793–9803. URL https://aclanthology.org/2025.coling-main.654/.

[37] Safdarian, A., Mohammadi, M., Jahanbakhsh, E., Naderi, M.S. and Faili, H. (2025) Schemagraphsql: Efficient schema linking with pathfinding graph algorithms for text-to-sql on large-scale databases. *CoRR* **abs/2505.18363**. doi:10.48550/ARXIV.2505.18363, URL https://doi.org/10.48550/arXiv.2505.18363. 2505.18363.

[38] Cao, R., Chen, L., Chen, Z., Zhao, Y., Zhu, S. and Yu, K. (2021) LGESQL: line graph enhanced text-to-sql model with mixed local and non-local relations. In Zong, C., Xia, F., Li, W. and Navigli, R. [eds.] *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021* (Association for Computational Linguistics): 2541–2555. doi:10.18653/V1/2021.ACL-LONG.198, URL https://doi.org/10.18653/v1/2021.acl-long.198.

[39] Wang, P., Sun, B., Dong, X., Dai, Y., Yuan, H., Chu, M., Gao, Y. *et al.* (2025) Agentar-scale-sql: Advancing text-to-sql through orchestrated test-time scaling. *CoRR* **abs/2509.24403**. doi:10.48550/ARXIV.2509.24403, URL https://doi.org/10.48550/arXiv.2509.24403. 2509.24403.

[40] Li, X., You, J., Li, H., Peng, J., Chen, X. and Guo, Z. (2025) CM-SQL: A cross-model consistency framework for text-to-sql. *Neurocomputing* **658**: 131708. doi:10.1016/J.NEUCOM.2025.131708, URL https://doi.org/10.1016/j.neucom.2025.131708.

[41] Ma, X., Tian, X., Wu, L., Wang, X., Tang, X. and Wang, J. (2024) Enhancing text-to-sql capabilities of large language models via domain database knowledge injection. In Endriss, U., Melo, F.S., Bach, K., Diz, A.J.B., Alonso-Moral, J.M., Barro, S. and Heintz, F. [eds.] *ECAI*

2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024) (IOS Press), *Frontiers in Artificial Intelligence and Applications* **392**: 3859–3866. doi:10.3233/FAIA240949, URL https://doi.org/10.3233/FAIA240949.

[42] Zhan, Z., E, H. and Song, M. (2025) Leveraging large language model for enhanced text-to-sql parsing. *IEEE Access* **13**: 30497–30504. doi:10.1109/ACCESS.2025.3540072, URL https://doi.org/10.1109/ACCESS.2025.3540072.

[43] Zhong, V., Xiong, C. and Socher, R. (2017) Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR* **abs/1709.00103**. URL http://arxiv.org/abs/1709.00103. 1709.00103.

[44] Chu, Z., Wang, Z. and Qin, Q. (2024) Leveraging prior experience: An expandable auxiliary knowledge base for text-to-sql. *CoRR* **abs/2411.13244**. doi:10.48550/ARXIV.2411.13244, URL https://doi.org/10.48550/arXiv.2411.13244. 2411.13244.

[45] Li, C., Shao, Y. and Liu, Z. (2024) SEA-SQL: semantic-enhanced text-to-sql with adaptive refinement. *CoRR* **abs/2408.04919**. doi:10.48550/ARXIV.2408.04919.

[46] Cen, J., Liu, J., Li, Z. and Wang, J. (2025) Sqlfixagent: Towards semantic-accurate text-to-sql parsing via consistency-enhanced multi-agent collaboration. In Walsh, T., Shah, J. and Kolter, Z. [eds.] *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025* (Philadelphia, PA, USA: AAAI Press): 49–57. doi:10.1609/AAAI.V39I1.31979.

[47] Zhang, B., Ye, Y., Du, G., Hu, X., Li, Z., Yang, S., Liu, C.H. *et al.* (2024) Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *CoRR* **abs/2403.02951**. doi:10.48550/ARXIV.2403.02951.

[48] Li, H., Wu, S., Zhang, X., Huang, X., Zhang, J., Jiang, F., Wang, S. *et al.* (2025) Omnisql: Synthesizing high-quality text-to-sql data at scale. *CoRR* **abs/2503.02240**. doi:10.48550/ARXIV.2503.02240.

[49] Maamari, K., Abubaker, F., Jaroslawicz, D. and Mhedhbi, A. (2024) The death of schema linking? text-to-sql in the age of well-reasoned language models. *CoRR* **abs/2408.07702**. doi:10.48550/ARXIV.2408.07702.

[50] Caferoglu, H.A. and Ulusoy, Ö. (2024) E-SQL: direct schema linking via question enrichment in text-to-sql. *CoRR* **abs/2409.16751**. doi:10.48550/ARXIV.2409.16751.