CoGSQL: A Cognitive Framework for Enhancing Large Language Models in Text-to-SQL Translation

Hongwei Yuan[1, 2], **Xiu Tang**[1, 2], **Ke Chen**[1, 2], **Lidan Shou**[1, 2], **Gang Chen**[1, 2], **Huan Li**[1, 2*]

[1]The State Key Laboratory of Blockchain and Data Security, Zhejiang University
[2]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security
yhw.cs, lihuan.cs@zju.edu.cn

Abstract

Large language models (LLMs) have significantly advanced the performance of various natural language processing tasks, including text-to-SQL. Current LLM-based text-to-SQL schemes mainly focus on improving the understanding of natural language questions (NLQs) or refining the quality of generated SQLs. While these strategies are effective, they often address specific, nuanced aspects. In contrast, humans approach text-to-SQL with a holistic view, applying transitional logical reasoning across multiple steps to arrive at the final answer. We believe LLMs can leverage human cognitive processes to achieve greater accuracy in text-to-SQL. In this paper, we present CoGSQL, a framework featuring a suite of tailored models and strategies aimed at replicating human cognitive processes for enhanced LLM-based text-to-SQL. CoGSQL consists of three key modules: (1) SQL preparation: we employ a coarse-to-fine schema linking and syntax keyword prediction, akin to how human recall and align key concepts for better understanding. (2) SQL generation: we introduce a concept-enhanced chain-of-thought prompting, enhancing NLQ interpretation and SQL composition of LLMs, similar to humans drafting SQL query. (3) SQL correction: we develop NLQ consistency and result consistency techniques to correct various errors, mirroring how humans evaluate and refine reasoning. We conduct extensive experiments using diverse benchmarks and LLMs. The results and analysis verify the effectiveness and generalizability of CoGSQL.

Extended version —
https://github.com/Yhw109/COGSQL

Introduction

Relational databases store large amounts of data, but Structured Query Language (SQL), the tool used to retrieve data from these databases, has traditionally been a skill limited to expert users. Translating natural language questions (NLQs) into SQL queries, known as text-to-SQL, has been a research focus for years (Zelle and Mooney 1996; Saha et al. 2016; Yu et al. 2018). This is crucial for creating more user-friendly database interfaces.

Recently, phenomenal large language models (LLMs) have excelled in various tasks such as question answering (Nguyen et al. 2023) and math reasoning (Zhang et al. 2023b). They have also significantly improved the capabilities of text-to-SQL tasks. Directly instructing LLMs, whether open-source or proprietary, has become a favorable paradigm. This approach avoids costly fine-tuning, easily transitions between LLM products, and benefits from their performance upgrades. When we mention LLM-based text-to-SQL, it refers to this inference-only approach. Current studies in this area aim to improve understanding of NLQs or enhance the quality of generated SQLs. For example, (Pourreza and Rafiei 2024; Xie et al. 2024) use task decomposition to enhance LLMs' attention towards simple subtasks; (Gao et al. 2024; Ren et al. 2024) employ retrieval-based few-shot demonstrations to elicit LLMs' in-context learning ability; (Qu et al. 2024) propose task alignment to mitigate LLMs' hallucinations in SQL generation; and (Wang et al. 2023a) propose multi-agent collaboration to jointly enhance text-to-SQL quality. While these strategies improve performance, they often concentrate on specific, nuanced aspects. In contrast, humans approach text-to-SQL with a holistic view, using reasoning across various aspects to achieve accurate and dependable translations. This cognitive paradigm involves making connections and transitions across different steps to ensure precision and reliability.

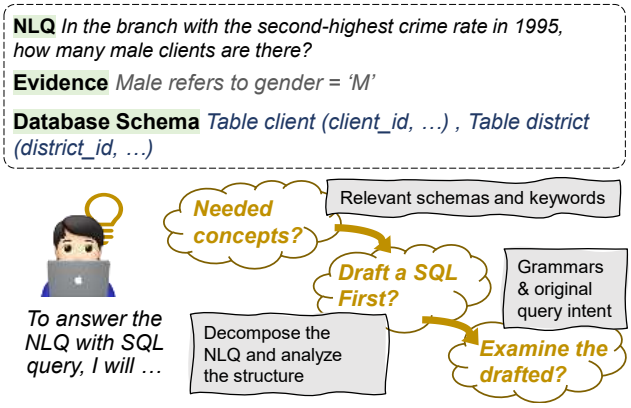

Figure 1: Illustration of addressing a text-to-SQL case following a human cognitive process of three stages.

Figure 1 depicts how a human applies transitional logical thinking to transform text into SQLs in an integrated

*Huan Li is the corresponding author.

manner.[1] When presented with the NLQ, a person begins by identifying essential concepts, focusing on crucial parts of problem-solving, like relevant database schemas and syntax keywords. Next, they draft the SQL using these key concepts. Usually, before writing down the SQL, they analyze its structure via decomposition, paying attention to detailed SQL clauses and subqueries, and then compose the SQL. Finally, they review the drafted SQL, considering aspects like grammar and alignment with the original NLQ intent, making necessary corrections.

This thus leads us to a critical question: *Can LLMs benefit from humans' cognitive process to improve text-to-SQL accuracy?* We believe the answer is **yes**. In this paper, we present COGSQL, a framework designed to imitate human **COG**nitive process to enhance LLMs in text-to-**SQL**. Based on the stages in Figure 1, COGSQL is divided into three main modules: (i) SQL PREPARATION, which identifies relevant database schemas and syntax keywords, akin to recalling key concepts; (ii) SQL GENERATION, which enables LLMs to interpret NLQs like humans and construct SQL precisely via concept-enhanced chain-of-thought (CoT) prompting; (iii) SQL CORRECTION, which instructs LLMs to reflect on the generated SQL, correcting grammar errors and aligning with NLQ intent. Each module includes tailored models or strategies to ensure LLMs deliver high-quality responses by following this cognitive process.

We perform experiments on five large challenging cross-domain text-to-SQL datasets with different LLMs, including both robust proprietary and fine-tuned open-source models. Results verify that COGSQL effectively enhances the performance of LLM-based text-to-SQL and generalizes well across various LLMs. Our main contributions include

1. We propose a novel text-to-SQL framework that aims to holistically enhance LLM's text-to-SQL abilities by imitating human cognitive process.
2. We examine our framework on five cross-domain text-to-SQL benchmarks with various LLMs, showing its effectiveness and generalization across datasets and models.

Related Work

Text-to-SQL Progress. Translating NLQs into SQLs has long been a focus in database and NLP research. Early approaches (Zelle and Mooney 1996; Simitsis, Koutrika, and Ioannidis 2008; Li and Jagadish 2014) relied on labor-intensive, manually crafted rules. The advent of neural networks has popularized the encoder-decoder architecture (Li and Jagadish 2014; Saha et al. 2016; Wang et al. 2020), where the encoder processes the NLQ and database schema, and the decoder generates the SQL query. Transformer-based architectures and pre-trained language models (Scholak, Schucher, and Bahdanau 2021; Li et al. 2023a,b) have further improved text-to-SQL performance. Recently, LLMs have demonstrated exceptional capabilities, with inference-only approaches (Dong et al. 2023; Wang et al. 2023a; Gao et al. 2024; Qu et al. 2024; Xie et al. 2024; Ren et al. 2024) offering superior transferability.

SQL Preparation. The LLM's generation process is enhanced by gathering critical information and performing NLQ-schema linking. Schema linking can be coarse-grained using cross-encoders (Li et al. 2023a, 2024b) or fine-grained through demonstrations and instructions (Pourreza and Rafiei 2024; Wang et al. 2023a; Dong et al. 2023), improving translation accuracy. Additionally, demonstration selection, which optimizes the LLM's in-context learning by choosing NLQ-SQL examples, is crucial. While retrieval-based selection (Chang and Fosler-Lussier 2023; Gao et al. 2024; Pourreza and Rafiei 2024; Ren et al. 2024) is common but computationally expensive, COGSQL streamlines this process simply by using a fixed two-shot demonstration in prompts.

SQL Generation. SQL generation involves complex reasoning that builds on key information from SQL preparation. LLMs excel in these tasks when effectively prompted. Techniques like chain-of-thought (CoT) (Wei et al. 2024; Kojima et al. 2024), decomposition and planning (Zhou et al. 2023; Wang et al. 2023b), and tool incorporation (Gao et al. 2023; Chen et al. 2024) greatly enhance LLMs' generation. Studies (Zhang et al. 2023a; Tai et al. 2023; Pourreza and Rafiei 2024) have applied CoT specialized to text-to-SQL. For instance, (Pourreza and Rafiei 2024) use a two-stage process where the LLM assesses question difficulty and applies CoT for complex queries, while (Zhang et al. 2023a; Tai et al. 2023) perform schema mapping before generating SQL. Differently, COGSQL integrates NLQ interpretation and SQL structure analysis for in-depth reasoning.

SQL Correction. Error correction for generated SQLs is achieved through techniques like self-consistency (Dong et al. 2023; Gao et al. 2024; Ren et al. 2024), self-correction (Pourreza and Rafiei 2024), and verification via execution results (Ni et al. 2023) to ensure reliability. However, inappropriate self-correction can reduce accuracy (Pourreza and Rafiei 2024), and self-consistency can be computationally expensive due to repeated LLM inference. COGSQL mitigates these challenges by implementing a multi-view correction, which aims to align with both NLQ and execution results.

Preliminaries

Definition. Let an NLQ be $\mathcal{Q} = \{q_1, \ldots, q_{|\mathcal{Q}|}\}$, where each q_i is a word token, and a corresponding database schema be $\mathcal{D} = \langle \mathcal{T}, \mathcal{C}, \mathcal{R} \rangle$, where:

- $\mathcal{T} = \{t_1, \ldots, t_n\}$, with each t_i representing a table in the database. We use $|t_i|$ to denote t_i's column count;
- $\mathcal{C} = \{c_1^1, \ldots, c_1^{|t_1|}, c_2^1, \ldots, c_2^{|t_2|}, \cdots, c_n^1, \cdots, c_n^{|t_n|}\}$, with each c_i^j representing the j-th column in the i-th table;
- $\mathcal{R} = \{(c_k^i, c_h^j) \in \mathcal{C}^2\}$, where each pair (c_k^i, c_h^j) denotes a foreign key relationship between columns c_k^i and c_h^j.

The goal of text-to-SQL is to convert $\mathcal{Q}$ into an executable SQL statement on $\mathcal{D}$ to retrieve the desired results.

Key Concepts for Solving Text-to-SQL. Mimicking human cognitive processes in problem-solving involves recall-

[1]Humans can use different strategies to write SQL, and our COGSQL proposal stems from a preliminary study where we surveyed Computer Science graduate students.

ing *key concepts* to clarify the problem-solving direction and arrive at the correct answer. For prompting LLMs to generate SQL statements from NLQs, key concepts include: 1) *database items* related to the NLQ, such as tables, columns, and values; 2) *syntax keywords* needed to construct SQL, including clauses, functions, and operators. We categorize syntax keywords in Table 1 based on their practical applications. Not all SQL syntax keywords are enumerated here; we exclude those not present in our training datasets. This focus ensures our prediction is tailored to SQL generation while allowing for the integration of new keywords as needed.

Category Level	Syntax Keywords
Clause	SELECT, FROM, WHERE, ORDER BY, GROUP BY, HAVING
Aggregation	COUNT, SUM, AVG, MIN, MAX
String	SUBSTR
Date	STRFTIME
Conditional	CASE WHEN, IIF
null-related	IS NULL, IS NOT NULL
Comparison	IN, BETWEEN, LIKE
Arithmetic	Division ('/')
Other keywords	DISTINCT, CAST

Table 1: Categorization of syntax keywords to utilize.

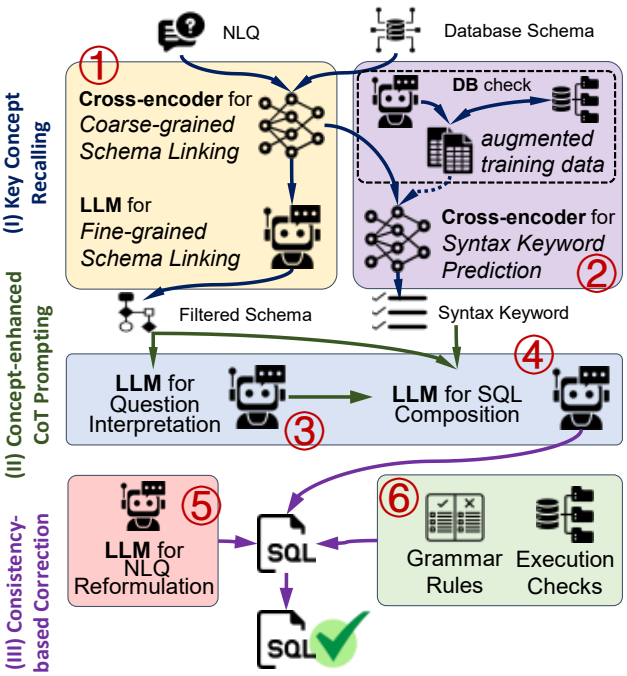

Figure 2: Overview of the proposed CoGSQL.

Methodology

Overview

Figure 2 provides an overview of the proposed CoGSQL, which enhances LLM-based text-to-SQL by emulating human cognitive processes through three key modules: (I) **Key Concept Recalling**, which performs ① *coarse-to-fine schema linking* and ② *syntax keyword prediction* to identify filtered schema and syntax keywords from the given NLQ and schema, respectively; (II) **Concept-enhanced CoT Prompting**, which employs a two-stage prompting method with ③ *NLQ interpretation* and ④ *SQL composition*, allowing LLMs to utilize key concepts to draft intermediate plans before finalizing the answer. (III) **Consistency-based Correction**, which enables LLMs to re-examine their answers for ⑤ *NLQ consistency* and ⑥ *result consistency* and make necessary adjustments.

Key Concept Recalling

Recalling key concepts initiates human cognitive processes for complex reasoning tasks. Before formulating SQLs, humans typically focus on relevant database items and syntax keywords. Accordingly, we propose *coarse-to-fine schema linking* and *syntax keyword prediction*.

Coarse-to-fine Schema Linking. Real-world databases often contain numerous tables and columns in their schema, which can overwhelm an LLM and hinder SQL generation due to length constraints. Schema linking (Pourreza and Rafiei 2024; Li et al. 2023a) aims to identify a subset of the schema. It should be *comprehensive* to include all necessary database items (i.e., needed tables in $\mathcal{T}$ and columns in $\mathcal{C}$) for the LLM to formulate SQLs and *concise* to avoid extra tables and columns that can mislead the LLM's generation.

Current LLM-based text-to-SQL studies primarily use two strategies. One (Qu et al. 2024; Pourreza and Rafiei 2024; Wang et al. 2023a) prompts LLMs directly to select relevant tables and columns, while another treats schema linking as a classification task (Li et al. 2023a, 2024b):

$$\mathcal{Y}^{\text{sl}} = f(\text{Enc}(\mathcal{Q}, \mathcal{T}, \mathcal{C}); \theta^{\text{sl}}), \quad (1)$$

where an encoder $\text{Enc}(\cdot)$ maps the input sequence into embeddings with Transformer-based architectures, typically RoBERTa (Liu et al. 2019), and a classifier $f(\cdot;\theta)$, parameterized by θ, outputs probabilities for each database item.

The encoder input consists of a concatenation of the NLQ $\mathcal{Q}$ with flattened table and column names. Embeddings for each table $t_i \in \mathcal{T}$ and column $c_j \in \mathcal{C}$ are derived from the final layer hidden state of the encoder. The classifier then produces probabilities for n tables and all $(m = \sum_{i=1}^{n} |t_i|)$ columns, represented as $\mathcal{Y}^{\text{sl}} = y_1^t, \ldots, y_n^t, y_1^c, \ldots, y_m^c$, where y_i^t denotes the probability of the i-th table and y_j^c denotes the probability of the j-th column. At the table level, the top-k_1 tables with the highest probabilities are retained; similarly, for each selected table, the top-k_2 columns with the highest probabilities are retained.

The first strategy is simple to implement and can produce a minimized schema. However, it risks omitting critical tables and columns, as the LLM may overlook subtle semantic nuances in the NLQ. The second strategy can retain all necessary database items by maintaining a relatively expansive schema (in comparison to the former strategy), but can introduce more noise. In this light, CoGSQL proposes a coarse-to-fine schema linking approach that combines their advantages. Following Eq. 1, it first uses a classifier for *coarse-grained* schema linking to filter out unnec-

essary tables and columns. Then, it prompts the LLM with the retained schema (k_1 tables and $k_1 \times k_2$ columns) and refines it by selecting a subset of tables (say k_3) out of the k_1 tables for *fine-grained* schema linking. To improve the overall efficiency of CoGSQL, this selection procedure is integrated with the first stage of concept-enhanced CoT prompting (see Figure 3 for a simplified version of our prompt and LLM's response). We only perform fine-grained schema linking at the table level to avoid column omission. The filtered schema in CoGSQL consists of k_3 tables, their retained columns, and corresponding foreign keys in $\mathcal{R}$.

Syntax Keyword Prediction.

We treat syntax keyword prediction as a classification task similar to the coarse-grained scheme linking in Eq. 1:

$$\mathcal{Y}^{\text{kw}} = f(\text{Enc}(\mathcal{I}, \mathcal{Q}, \hat{\mathcal{T}}, \hat{\mathcal{C}}); \theta^{\text{kw}}). \quad (2)$$

Here, the encoder input concatenates a fixed instruction $\mathcal{I}$ (e.g., "*Determine whether the SQL corresponding to the following NLQ need to use ORDER BY*"), $\mathcal{Q}$, and filtered tables $\hat{\mathcal{T}}$ and columns $\hat{\mathcal{C}}$ from coarse-grained schema linking — this reduces noise compared to using the entire schema.

The `[cls]` token embedding from the final-layer hidden state of the encoder, is fed into different classifiers of the same structure to obtain the probability corresponding to specific syntax keywords listed in Table 1. A threshold τ is used to decide whether a syntax keyword is needed for the given NLQ. We perform differentiated predictions for different keywords based on their distribution in the training set. We do not predict ubiquitous keywords like `SELECT`, `FROM`, and `WHERE`. Dedicated classifiers are used separately for `ORDER BY`, `GROUP BY`, and `HAVING`. For remaining keywords, classifiers are built per category level (see Table 1); e.g., we predict the need for the aggregation functions rather than directly predicting an individual keyword `SUM`.

However, the imbalanced distribution of syntax keywords (e.g., the rare appearance of `HAVING`) can hinder the performance of the classifiers. To mitigate this, text-to-SQL augmentation methods (Yu et al. 2021; Hu et al. 2023) have been proposed, which generate valid SQL queries from extracted SQL templates. However, these methods cannot ensure alignment with the original training distribution.

Inspired by a recent study (Yu et al. 2024), we propose ***sample-centric augmentation***, leveraging LLMs to synthesize high-quality NLQ-SQL pairs from original training samples. For a given training sample ($\mathcal{Q}$, $\mathcal{S}$), this method offers two augmentations: 1) *NLQ Rewriting*: if $\mathcal{S}$ contains the keyword to augment, LLMs formulate a new NLQ $\mathcal{Q}'$ based on $\mathcal{S}$, preserving $\mathcal{Q}$'s intent; e.g., when augmenting `COUNT`, the NLQ in Figure 1 is rewritten as "*Considering the branch where the second-highest number of crimes occurred in 1995,* ***how many of its clients*** *are male?*". 2) *Keyword Addition*: otherwise, LLMs are prompted to add the keyword to augment $\mathcal{S}$ and formulate a corresponding NLQ, producing ($\mathcal{S}'$, $\mathcal{Q}'$); e.g., when augmenting `ORDER BY`, the NLQ in Figure 1 becomes "*In the branch where the second-highest number of crimes were committed in 1995 occurred, which male client has a* ***latest*** *birthday?*" *—* ***latest*** implies added `ORDER BY`. These augmentations enrich the keyword distribution of the training set while preserving semantic integrity. Moreover, we adapt a high temperature to enrich generation diversity. We use execution-based checks for correctness: LLMs are prompted to answer $\mathcal{Q}'$ with $\mathcal{S}''$; NLQ rewriting should ensure $\mathcal{S}$ and $\mathcal{S}''$ produce identical execution results; keyword addition must ensure $\mathcal{S}'$ and $\mathcal{S}''$ yield identical and *non-empty* results.

Prompt
Instruction: Read the question to identify the required data to query and return. Assess if subqueries are needed. **Database Schema**: Table client (client_id, …) , Table district(district_id, …) **Question**: In the branch with the second-highest crime rate in 1995, how many male clients are there?
LLM response
To Be Queried: ['the 2nd-highest number of crimes'] **To Return**: ['the number of male clients'] **Table Needed**: ['client', 'district'] **Subquery Required**: ['yes']

Figure 3: Prompt and LLM response for NLQ interpretation.

Concept-enhanced CoT Prompting

To improve SQL generation with LLMs, we address two crucial issues, interpreting NLQ intents and composing SQL queries. We incorporate solutions into a CoT prompting process, mirroring how humans mentally rehearse key concepts before responding formally. This two-stage prompting involves *NLQ interpretation* that identifies inquiry objectives and desired data and *SQL composition* that ensures the correct use of identified key concepts.

NLQ Interpretation. Referring to the example NLQ in Figure 1. Upon seeing this NLQ, humans first analyze its content to identify the inquiry objective ("second-highest number of crimes committed in 1995") and returned data ("number of male clients"). For complex NLQs with nested query (subquery and set operation), such as the NLQ in Figure 1, it is also crucial to decide if such a query is needed. An illustrative prompt that enables such a procedure is provided in Figure 3, which includes a specific instruction to guide LLMs, the schema after coarse-grained schema linking, and the NLQ itself. To ensure the LLM reasons effectively, we provide demonstrations of two NLQs manually annotated[2] from the training set.

The LLM response is also exemplified in Figure 3, including the inquiry objective, the returned data, the table needed, and any required subqueries.

SQL Composition. Back to the human cognitive procedure: after interpreting an NLQ, humans identify how key concepts fit into various SQL clauses, such as identifying the filter condition for the `WHERE` clause. To replicate this procedure, we focus on decomposing and generating the SQL.

[2]NLQ-specific demonstration selection is for future work.

Prompt
Instruction: Analyze the intent of SQL clauses and subqueries, then formulate a SQL query. **Database Schema**: Table client (client_id, …) , Table district (district_id, …) … **Question**: In the branch with the second-highest crime rate in 1995 … **Keyword Indicator**: The SQL query is likely to need: FROM, WHERE, SELECT, COUNT …
LLM response
Clause Analysis: FROM clause for retrieving … WHERE clause for filtering … **Subquery Analysis**: Subquery for retrieving crime … **SQL**: SELECT count(client.client_id) FROM …

Figure 4: Prompt and LLM response for SQL composition.

We instruct LLM to analyze the SQL clauses identified by our syntax keyword prediction, decompose the NLQ, and generate it based on this analysis. An example prompt is provided in Figure 4. It includes demonstrations, a specific instruction, the database schema after fine-grained schema linking, NLQ, interpretation from the first stage, and a keyword indicator that includes the hints derived from syntax keyword predictions. The keyword indicator has two parts: 1) a clause list shows likely clauses and 2) operator indicator that maps likely operators to text hints. For example, if the classifier predicts an aggregation function category (see Table 1), operator indicator suggests "*The SQL is likely to involve at least one aggregation function: COUNT, SUM, AVG, MIN, MAX*". The LLM's response in Figure 4 involves clause analysis, subquery analysis, and the generated SQL. The above two-stage CoT ensures LLMs thoroughly consider recalled key concepts, effectively using them to gradually compose the desired SQL. This is beneficial for complex NLQs as our experiments show.

Consistency-based Correction

LLMs are known to experience mistakes and hallucinations (Ji et al. 2023) in generation. Specifically, two common mistakes could hamper LLMs' text-to-SQL translation: 1) *NLQ Misunderstanding*: LLMs may misinterpret the intent of an NLQ, leading to a semantically incorrect SQL query. 2) *Grammar Mistakes*: LLMs might violate SQL grammar rules, resulting in a non-executable query. To address these issues, CogSQL performs a consistency-based correction, incorporating *NLQ consistency* that ensures the SQL's alignment with the intended NLQ, and *result consistency* that ensures the SQL complies with grammatical rules.

NLQ Consistency. Some NLQs contain subtle semantic details. Ignoring these can result in LLMs' misunderstanding. For example, the NLQ in Figure 1 might ask for the number of male clients, but an LLM might incorrectly return both the count and their personal information. CogSQL performs

Prompt
Instruction: Formulate a new question to verify the correctness of SQL query. **Database Schema**: Table client (client_id, …) … **Question**: In the branch with the second-highest crime rate in … **SQL:** SELECT count(client.client_id) FROM …
LLM response
New NLQ: How many male clients are in the ... **Consistency with given NLQ:** Yes **SQL**: SELECT count(client.client_id) FROM …

Figure 5: Prompt and LLM response for NLQ-consistency.

NLQ-consistency correction to address this issue. Examples of a simplified version of our prompt and LLM's response are in Figure 5. We instruct LLMs to formulate a new NLQ based on the schema and generated SQL, and then verify if this new NLQ is consistent with the original one. If they match, the LLM has correctly understood the original NLQ. If not, the SQL query likely needs revision. This revision process is formulated as:

$$\hat{\mathcal{Q}}, \mathcal{C}, \hat{\mathcal{S}} = f_{\mathcal{M}}(\mathcal{I}, \mathcal{D}, \mathcal{Q} \mid \mathcal{S}), \tag{3}$$

where $f_{\mathcal{M}}(\cdot \mid \cdot)$ is a mapping function applied by the LLM $\mathcal{M}$, $\mathcal{I}$ is the instruction, $\mathcal{D}$ is the schema, $\mathcal{Q}$ is the original NLQ, and $\mathcal{S}$ is the generated SQL before correction. $\hat{\mathcal{Q}}$ is the new NLQ formulated for $\mathcal{S}$, $\mathcal{C}$ is the consistency check result, and $\hat{\mathcal{S}}$ is the SQL after potential correction. This correction is done without needing real-world database access, and we apply it to all generated SQLs.

Result Consistency. SQL language has complex grammar rules, and violating these can result in non-executable SQL queries. CogSQL performs result-consistency correction to ensure that generated SQL queries are grammatically correct. This correction consists of two aspects. First, we establish correction rules (e.g., if a column name in the SQL query contains spaces, it must be enclosed in double quotes), each generated SQL query undergoes a grammar check to ensure compliance with the established rules. The rule set is easily to be updated and expanded.

The SQL queries after rule-based correction are executed in databases. Non-executable ones will trigger exceptions during execution. We prompt LLMs to correct these non-executable SQL queries using the corresponding exceptions. This process can be formally expressed as:

$$\hat{\mathcal{S}} = f_{\mathcal{M}}(\mathcal{I}, \mathcal{D}, \mathcal{Q} \mid \mathcal{S}, \mathcal{E}), \tag{4}$$

where $f_{\mathcal{M}}(\cdot \mid \cdot)$ is the LLM-enabled mapping function, $\mathcal{I}$ the instruction, $\mathcal{D}$ the schema, $\mathcal{Q}$ the original NLQ, $\mathcal{S}$ the SQL query generated by $\mathcal{M}$; in addition, $\mathcal{E}$ is the exception encountered when SQL query executes in the database, $\hat{\mathcal{S}}$ is the SQL after correction. This correction is applied only to each non-executable SQL.

Experiment

We address the following research questions through comprehensive experiments:

- *RQ1*. How does COGSQL's performance compare to existing methods across various LLMs and datasets?
- *RQ2*. What are the contributions of each COGSQL component to the final text-to-SQL translation accuracy?
- *RQ3*. How does COGSQL mitigate the mistakes that LLMs are prone to make in text-to-SQL translation?

Experimental Setup

Datasets. We evaluate COGSQL using the well-recognized text-to-SQL benchmarks: (1) Spider (Yu et al. 2018). (2) Spider's variants (Li et al. 2023a), Spider-DK, Spider-Realistic, Spider-Syn. (3) BIRD (Li et al. 2024c).

Baselines. We compare COGSQL to a wide range of state-of-the-art methods, including RESDSQL (Li et al. 2023a), CODES (Li et al. 2024b), DIN-SQL (Pourreza and Rafiei 2024), MAC-SQL (Wang et al. 2023a), TA-SQL (Qu et al. 2024), DAIL-SQL (Gao et al. 2024), DEA-SQL (Xie et al. 2024), and SUPER-SQL (Li et al. 2024a). Among these, RESDSQL and CODES use *meticulously fine-tuned* T5 (Raffel et al. 2020) and STARCODER (Li et al. 2023c) as their base models. Other baselines similar to COGSQL do not involve fine-tuning the base language model.

Metrics. Following previous studies (Qu et al. 2024; Xie et al. 2024), we use ***Execution Accuracy (EX)*** and ***Valid Efficiency Score (VES)*** as performance metrics. (1) EX checks if the predicted SQL produces the same execution results as the ground truth SQL on the database. (2) VES, the specialized metric from BIRD, measures efficiency of the correctly predicted SQL. The measurement unit for both metrics is %, and higher measures indicate better effectiveness/efficiency.

Implementation. We use a temperature of 0.7 for sample-centric augmentation, and 0 for other modules of COGSQL. Each sample in Spider and BIRD training sets is augmented once, resulting in training with 1,390 (*resp*. 1,441) samples from NLQ rewriting and 2,665 (*resp*. 2,833) samples from keyword addition. Two-shot prompts are used for augmentation, concept-enhanced CoT, and NLQ-consistency correction modules. We use (Li et al. 2024b)'s checkpoint for coarse-grained schema linking. The encoder and classifiers for syntax keyword prediction are trained on an A6000 GPU with a learning rate of 1e-6, using LoRA with rank $= 128$. Based on analyzing the training set of two benchmarks, we use $k_1 = 5$ and $k_2 = 7$ for coarse-grained schema linking.

We evaluate COGSQL on both proprietary and open-source LLMs. For proprietary LLMs, we use the GPT series: GPT4-`0613`, GPT-4o-`2024-05-13`, and GPT-4o-mini-`2024-07-18`. For open-source, we use DEEPSEEK-CODER-`V2-0724` (DeepSeek-AI et al. 2024).

Overall Performance (RQ1)

Baseline Comparison. Referring to Table 2, COGSQL enhances the GPT4 baseline effectively on both benchmarks. Even with modest models like GPT-4o-mini, COGSQL remains competitive, surpassing strong baselines like DIN-SQL and DAIL-SQL using GPT4. When equipped with stronger GPT-4o and GPT4, COGSQL outperforms extensively fine-tuned baselines RESDSQL and CODES. While COGSQL with GPT4 slightly lags behind SUPER-SQL and DEA-SQL on Spider test, those methods rely on well-designed criteria and exhaustive retrieval-based demonstration selection.

Note that Spider is more conventional, allowing in-context learning to effectively incorporate SQL knowledge via prompts. In contrast, the more challenging BIRD reduces the efficacy of demonstrations due to the complexity of NLQs, with both SUPER-SQL and DEA-SQL performing worse than COGSQL. This indicates that a holistic cognitive framework better enhances LLMs in complex scenarios. Nonetheless, COGSQL consistently ranks among the top two performers across all cases.

Moreover, BIRD's VES results also highlight COGSQL's ability to generate efficient SQL queries.

Methods	Spider		BIRD Dev	
	Dev EX	Test EX	EX	VES
RESDSQL-3B + NatSQL	84.10	79.90	\	\
SFT CODES-7B	85.40	\	57.17	58.80
SFT CODES-15B	84.90	\	58.47	59.87
GPT4	72.30	\	46.35	49.77
DIN-SQL + GPT4	83.50	85.30	50.72	58.79
MAC-SQL + GPT4	78.60	82.80	57.56	58.76
DAIL-SQL + GPT4	83.10	86.20	54.76	56.08
DEA-SQL + GPT4	85.40	**87.10**	58.93	63.07
TA-SQL + GPT4	85.00	\	56.19	\
SUPER-SQL + GPT4	**87.00**	\	58.50	61.99
COGSQL + GPT4	85.40	86.40	**59.58**	**64.30**
COGSQL + GPT-4o	84.70	85.80	59.19	63.99
COGSQL + GPT-4o-mini	84.20	83.80	56.26	61.31

Table 2: Overall comparison: Grey text indicates the methods extensively fine-tuned on LLMs. The best performance is **bolded** and the second-best is underlined.

Spider Variants. Table 3 presents COGSQL's accuracies on three Spider variants that better reflect real-world applications. We compare two closest competitors using GPT4, TA-SQL and DEA-SQL. COGSQL effectively enhances both the GPT-4 and GPT-4o-mini baselines across all benchmarks, demonstrating impressive performance when NLQs are more obscure and require domain knowledge. Moreover, the results indicate that COGSQL with GPT-4o-mini competes well with other strong baselines using GPT4, further demonstrating its robustness and effectiveness. We presume that the NLQ interpretation in concept-enhanced CoT prompting allows LLMs to achieve robust understanding, even when NLQs are unclear. This further proves the importance of equipping LLMs with a human-akin mindset to interpret NLQs and compose SQLs.

Various LLMs. We use the more complex BIRD dev set to examine COGSQL's generalizability across various LLMs. The NLQs of BIRD are manually annotated by the benchmark authors to reflect different difficulty levels. Referring to Table 4, COGSQL significantly boosts DEEPSEEK performance across all difficulty levels, indicating strong gener-

Methods	DK	Syn	Realistic
GPT4	65.2	71.1	73.4
TA-SQL + GPT4	72.9	\	79.5
DEA-SQL + GPT4	\	\	81.5
GPT-4o-mini	69.9	69.8	78.7
GPT-4o-mini + CoGSQL	**73.8** (↑ 3.9)	**76.0** (↑ 6.2)	**81.5** (↑ 2.8)
GPT-4 + CoGSQL	**76.6** (↑ 11.4)	**76.5** (↑ 5.4)	**85** (↑ 11.6)

Table 3: EX (%) results on three Spider variants.

alizability across models. We also present the improvement that CoGSQL brings to GPT-4o-mini, with the most significant gains in the 'challenging' category, followed by 'moderate'. This demonstrates that CoGSQL's effectiveness in enhancing LLMs' ability to reason through complex NLQs.

Model	BIRD Dev			
	Simple	Moderate	Challenging	Total
DeepSeek	61.62	44.73	29.17	53.46
+ CoGSQL	64.11 (↑ 2.49)	46.67 (↑ 1.94)	39.58 (↑ 10.41)	56.52 (↑ 3.06)
GPT-4o-mini	58.27	38.92	28.47	49.61
+ CoGSQL	63.24 (↑ 4.97)	47.31 (↑ 8.39)	40.28 (↑ 11.81)	56.26 (↑ 6.65)

Table 4: EX (%) results on both proprietary and open-source LLMs; (↑) indicates the performance gain.

Ablation Study (RQ2)

Module Design. Referring to Table 5, every module contributes to overall performance, with any removal causing declines. On Spider, CoGSQL exhibits robustness, maintaining stable performance even when individual modules are removed, except for the coarse-to-fine schema linking module. Its absence causes the most significant drop, showing the importance of accurate schema linking. The NLQ consistency and result consistency modules are equally effective for Spider. For more complex BIRD, the concept-enhanced CoT prompting and result consistency modules are vital. Removing them leads to the largest performance drops, underscoring their importance in handling BIRD's complex schema and reasoning demands.

Methods	Spider	Bird
CoGSQL + GPT-4o-mini	**84.20**	**56.26**
- w/o Coarse-to-fine Schema Linking	82.40	55.74
- w/o Syntax Keyword Prediction	83.50	55.02
- w/o Concept-enhanced CoT Prompting	83.50	52.41
- w/o NLQ Consistency	83.70	55.93
- w/o Result Consistency	83.60	53.85

Table 5: Ablations of CoGSQL modules using EX (%).

Augmentation Strategies. Table 6 shows that our augmentations significantly improve classifier performance in syntax keyword prediction. Keyword addition proves more beneficial than NLQ rewriting by boosting data diversity.

Model Variant	Total AUC
CoGSQL classifiers	**10.0681**
- w/o NLQ rewriting	9.9918
- w/o keywords addition	9.9663
- w/o both augmentations	9.8483

Table 6: Ablations of augmentation strategies on BIRD. 'Total AUC' refers to the sum of AUCs of all trained classifiers.

Fine-grained Analysis (RQ3)

To scrutinize the impact of CoGSQL, we conduct an error analysis, comparing mistakes made by GPT-4o-mini baseline to those made by GPT-4o-mini + CoGSQL. We classify LLM mistakes into five main categories: 1) *Schema Misuse*: Incorrect use or omission of table and column names in SQLs. 2) *Keyword Misuse*: Incorrect use of SQL keywords. 3) *Nested Query Misuse*: Failure to recognize the need for, or inappropriate use of, nested queries. 4) *NLQ Misunderstanding*: Misinterpretation of the NLQ intent. 5) *Syntax Error*: Generation of non-executable SQLs due to grammar errors.

We randomly select 500 NLQs from BIRD and analyze the distribution of mistakes, and explore how CoGSQL mitigate them. Schema misuse is the most common error in Table 7, showing the difficulty of accurately identifying necessary tables and columns amidst complex schemas and NLQs. Keyword misuse and nested query misuse are closely tied to NLQ's complexity, indicating a need for improved reasoning in complex SQL queries. CoGSQL effectively reduces errors in all categories, proving its ability to follow cognitive processes to resolve text-to-SQL progressively.

Mistake Category	GPT-4o-mini	GPT-4o-mini+CoGSQL
Schema Misuse	123	75 (↓ 48)
Keyword Misuse	46	35 (↓ 11)
Nested Query Misuse	47	39 (↓ 8)
NLQ Misunderstanding	35	22 (↓ 13)
Syntax Error	11	2 (↓ 9)
Total	262	173 (↓ 89)

Table 7: Mistake analysis for 4o-mini baseline and 4o-mini enhanced with CoGSQL; (↓) indicates diminished mistakes.

Conclusion

While LLMs elevate the performance of text-to-SQL to the next level, complex benchmarks such as BIRD continue to challenge even the most advanced LLMs. In this study, we present CoGSQL, a novel framework designed to emulate human cognitive processes to enhance LLMs' reasoning abilities. CoGSQL leverages coarse-to-fine schema linking and syntax keyword prediction to effectively recall key concepts, followed by concept-enhanced CoT prompting for precise SQL generation. Consistency-based correction, from both NLQ and result perspectives, ensures robust and reliable adjustment to final outputs. Extensive experiments across diverse datasets and LLMs verify CoGSQL's superiority and confirm the benefits of mimicking human cognition.

Acknowledgments

We thank the anonymous reviewers for their helpful suggestions. This work was supported by the Pioneer R&D Program of Zhejiang (No.2024C01021), the Major Research Program of Zhejiang Provincial Natural Science Foundation (No. LD24F020015), and the NSFC Grant No. U24A201401.

References

Chang, S.; and Fosler-Lussier, E. 2023. Selective Demonstrations for Cross-domain Text-to-SQL. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 14174–14189. Association for Computational Linguistics.

Chen, X.; Lin, M.; Schärli, N.; and Zhou, D. 2024. Teaching Large Language Models to Self-Debug. In *12th International Conference on Learning Representations, ICLR 2024*.

DeepSeek-AI; Zhu, Q.; Guo, D.; Shao, Z.; Yang, D.; Wang, P.; Xu, R.; Wu, Y.; Li, Y.; Gao, H.; Ma, S.; Zeng, W.; Bi, X.; Gu, Z.; Xu, H.; Dai, D.; Dong, K.; Zhang, L.; Piao, Y.; Gou, Z.; Xie, Z.; Hao, Z.; Wang, B.; Song, J.; Chen, D.; Xie, X.; Guan, K.; You, Y.; Liu, A.; Du, Q.; Gao, W.; Lu, X.; Chen, Q.; Wang, Y.; Deng, C.; Li, J.; Zhao, C.; Ruan, C.; Luo, F.; and Liang, W. 2024. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence. arXiv:2406.11931.

Dong, X.; Zhang, C.; Ge, Y.; Mao, Y.; Gao, Y.; Chen, l.; Lin, J.; and Lou, D. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306.

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. In *Proceedings of the VLDB Endowment, Volume 17, Issue 5*, 1132–1145.

Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; and Neubig, G. 2023. PAL: program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, 10764–10799. JMLR.org.

Hu, Y.; Zhao, Y.; Jiang, J.; Lan, W.; Zhu, H.; Chauhan, A.; Li, A. H.; Pan, L.; Wang, J.; Hang, C.-W.; Zhang, S.; Guo, J.; Dong, M.; Lilien, J.; Ng, P.; Wang, Z.; Castelli, V.; and Xiang, B. 2023. Importance of Synthesizing High-quality Data for Text-to-SQL Parsing. In *Findings of the Association for Computational Linguistics: ACL 2023*, 1327–1343. Association for Computational Linguistics.

Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y. J.; Madotto, A.; and Fung, P. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.*, 248:1–248:38.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2024. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 22199–22213. Curran Associates Inc.

Li, B.; Luo, Y.; Chai, C.; Li, G.; and Tang, N. 2024a. The Dawn of Natural Language to SQL: Are We Fully Ready? In *Proceedings of the VLDB Endowment, Volume 17, Issue 11*, 3318–3331.

Li, F.; and Jagadish, H. V. 2014. Constructing an interactive natural language interface for relational databases. In *Proceedings of the VLDB Endowment, Volume 8, Issue 1*, 73–84.

Li, H.; Zhang, J.; Li, C.; and Chen, H. 2023a. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, 13067–13075. AAAI Press.

Li, H.; Zhang, J.; Liu, H.; Fan, J.; Zhang, X.; Zhu, J.; Wei, R.; Pan, H.; Li, C.; and Chen, H. 2024b. CodeS: Towards Building Open-source Language Models for Text-to-SQL. In *Proceedings of the ACM on Management of Data, Volume 2, Issue 3*, 1–28.

Li, J.; Hui, B.; Cheng, R.; Qin, B.; Ma, C.; Huo, N.; Huang, F.; Du, W.; Si, L.; and Li, Y. 2023b. Graphix-T5: Mixing Pre-trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, 13076–13084. AAAI Press.

Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; Zhou, X.; Chenhao, M.; Li, G.; Chang, K.; Huang, F.; Cheng, R.; and Li, Y. 2024c. 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*, 42330–42357.

Li, R.; Allal, L. B.; Zi, Y.; Muennighoff, N.; Kocetkov, D.; Mou, C.; Marone, M.; Akiki, C.; Li, J.; Chim, J.; Liu, Q.; Zheltonozhskii, E.; Zhuo, T. Y.; Wang, T.; Dehaene, O.; Davaadorj, M.; Lamy-Poirier, J.; Monteiro, J.; Shliazhko, O.; Gontier, N.; Meade, N.; Zebaze, A.; Yee, M.-H.; Umapathi, L. K.; Zhu, J.; Lipkin, B.; Oblokulov, M.; Wang, Z.; Murthy, R.; Stillerman, J.; Patel, S. S.; Abulkhanov, D.; Zocca, M.; Dey, M.; Zhang, Z.; Fahmy, N.; Bhattacharyya, U.; Yu, W.; Singh, S.; Luccioni, S.; Villegas, P.; Kunakov, M.; Zhdanov, F.; Romero, M.; Lee, T.; Timor, N.; Ding, J.; Schlesinger, C.; Schoelkopf, H.; Ebert, J.; Dao, T.; Mishra, M.; Gu, A.; Robinson, J.; Anderson, C. J.; Dolan-Gavitt, B.; Contractor, D.; Reddy, S.; Fried, D.; Bahdanau, D.; Jernite, Y.; Ferrandis, C. M.; Hughes, S.; Wolf, T.; Guha, A.; von Werra, L.; and de Vries, H. 2023c. StarCoder: may the source be with you! arXiv:2305.06161.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692.

Nguyen, M. T.; Tran, K. T.; Nguyen, N. V.; and Vu, X.-S. 2023. ViGPTQA - State-of-the-Art LLMs for Vietnamese Question Answering: System Overview, Core Models Training, and Evaluations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, 754–764. Association for Computational Linguistics.

Ni, A.; Iyer, S.; Radev, D.; Stoyanov, V.; Yih, W.-t.; Wang, S. I.; and Lin, X. V. 2023. LEVER: learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, 26106–26128. JMLR.org.

Pourreza, M.; and Rafiei, D. 2024. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 36339–36348.

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 *Findings of the Association for Computational Linguistics: ACL 2024*, 5456—5471. Association for Computational Linguistics.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. 1–67.

Ren, T.; Fan, Y.; He, Z.; Huang, R.; Dai, J.; Huang, C.; Jing, Y.; Zhang, K.; Yang, Y.; and Wang, X. S. 2024. PURPLE: Making a Large Language Model a Better SQL Writer. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 15–28.

Saha, D.; Floratou, A.; Sankaranarayanan, K.; Minhas, U. F.; Mittal, A. R.; and Özcan, F. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. In *Proceedings of the VLDB Endowment, Volume 9, Issue 12*, 1209–1220.

Scholak, T.; Schucher, N.; and Bahdanau, D. 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*, 9895–9901. Association for Computational Linguistics.

Simitsis, A.; Koutrika, G.; and Ioannidis, Y. 2008. Précis: from unstructured keywords as queries to structured databases as answers. *The VLDB Journal*, 117–149.

Tai, C.-Y.; Chen, Z.; Zhang, T.; Deng, X.; and Sun, H. 2023. Exploring Chain of Thought Style Prompting for Text-to-SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 5376–5393. Association for Computational Linguistics.

Wang, B.; Ren, C.; Yang, J.; Liang, X.; Bai, J.; Chai, L.; Yan, Z.; Zhang, Q.-W.; Yin, D.; Sun, X.; and Li, Z. 2023a. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242.

Wang, B.; Shin, R.; Liu, X.; Polozov, O.; and Richardson, M. 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*, 7567–7578. Association for Computational Linguistics.

Wang, L.; Xu, W.; Lan, Y.; Hu, Z.; Lan, Y.; Lee, R. K.-W.; and Lim, E.-P. 2023b. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2609—2634. Association for Computational Linguistics.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q. V.; and Zhou, D. 2024. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 24824–24837.

Xie, Y.; Jin, X.; Xie, T.; Lin, M.; Chen, L.; Yu, C.; Cheng, L.; Zhuo, C.; Hu, B.; and Li, Z. 2024. Decomposition for Enhancing Attention: Improving LLM-based Text-to-SQL through Workflow Paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*, 10796—-10816. Association for Computational Linguistics.

Yu, L.; Jiang, W.; Shi, H.; Yu, J.; Liu, Z.; Zhang, Y.; Kwok, J. T.; Li, Z.; Weller, A.; and Liu, W. 2024. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. In *12th International Conference on Learning Representations, ICLR 2024*.

Yu, T.; Wu, C.-S.; Lin, X. V.; Wang, B.; Tan, Y. C.; Yang, X.; Radev, D.; Socher, R.; and Xiong, C. 2021. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. In *9th International Conference on Learning Representations, ICLR 2021*.

Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; Zhang, Z.; and Radev, D. 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*, 3911–3921. Association for Computational Linguistics.

Zelle, J. M.; and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2*, 1050–1055. AAAI Press.

Zhang, H.; Cao, R.; Chen, L.; Xu, H.; and Yu, K. 2023a. ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 3501–3532. Association for Computational Linguistics.

Zhang, M.; Wang, Z.; Yang, Z.; Feng, W.; and Lan, A. 2023b. Interpretable Math Word Problem Solution Generation via Step-by-step Planning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 6858–6877. Association for Computational Linguistics.

Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Cui, C.; Bousquet, O.; Le, Q.; and Chi, E. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *11th International Conference on Learning Representations, ICLR 2023*.