Structure Guided Large Language Model for SQL Generation

Anonymous ACL submission

Abstract

Generating accurate Structured Querying Language (SQL) is a long-standing problem, especially in matching users' semantic queries with structured databases and then generating structured SQL. Existing models typically input queries and database schemas into the LLM and rely on the LLM to perform semantic-structure matching and generate structured SQL. However, such solutions overlook the structural information within user queries and databases, which can be utilized to enhance the generation of structured SQL. This oversight can lead to inaccurate or unexecutable SQL generation. To fully exploit the structure, we propose the structure-to-SQL framework, which leverages the inherent structure information to improve the SQL generation of LLMs. Specifically, we introduce our **Structure Guided SQL (SGU-SQL)** generation model. SGU-SQL links user queries and databases in a structure-enhanced manner. It then decomposes complicated linked structures with grammar trees to guide the LLM to generate the SQL step by step. Extensive experiments on two benchmark datasets illustrate that SGU-SQL can outperform sixteen SQL generation baselines.

1 Introduction

Structured Querying Language (SQL) is the most popular language for communicating with databases and retrieving user-desired data. Due to the popularity of Large Language Models (LLMs), there is an increasing need to input a user query and automatically generate the corresponding SQL to communicate with the database. However, unlike natural languages, SQL is a standardized structured language that must exactly match the database in order to be executable and retrieve the correct data. Considering the impact of these factors, incorporating structure information can improve the performance of SQL generation models.

Existing state-of-the-art solutions primarily rely on the comprehensive and generative capabilities of LLMs to generate SQL, thus adopting a text-to-SQL approach. T5 (Raffel et al., 2020) first attempts to train a language model to understand the user query and build connections with the database. After the emergence of ChatGPT, especially GPT4 (OpenAI, 2023), some pioneers (Dong et al., 2023b; Ma et al., 2023) directly input the user query and database schema to ChatGPT to generate SQL. Building upon this, DIN-SQL (Pourreza and Rafiei, 2023) employs chain-of-thought by requiring ChatGPT to decompose the SQL generation into several sub-tasks and then lead ChatGPT to generate the SQL according to the sub-tasks.

However, existing models primarily rely on the semantic information of user queries and database schema, while overlooking the inherent structure of queries, databases, and SQLs. The neglect of structure results in the following limitations:

1. Semantic Ambiguity: Only relying on the semantics of the query and the database may not fully capture the user's intent. For example, when a user queries "income", it may ambiguously refer to the summation of the "salary" and "bonus" tables in the database. However, these tables are indirectly linked through the "ID" and "DT" (DateTime) keys. By adopting a purely semantic approach, there is a risk of overlooking this implicit linking and neglecting either the salary or the bonus.

2. Complex Relational Linking: Real-world databases often contain implicit and intricate structures, such as storing metadata in meta tables and storing transactional data in detail tables. The complicated structure presents challenges for semantic-based SQL generation in precisely organizing the complex structure.

3. Opaque Matching Processes: Existing models employ a black-box approach to query-database matching, which provides little interpretability to users. As a result, it is difficult for users to inspect

or refine the SQL generation process, thus limiting the reliability and usability.

While leveraging structural information to enhance SQL generation shows promise, it remains a challenging task due to the following reasons.
1. Structured Representation: The challenge lies in creating representations that accurately capture the structural nuances of queries and databases. It is especially demanding to develop a model that aligns query structures with database architectures.
2. Precise Query Matching: The key to SQL generation is linking query intents precisely with database structures. It is challenging to propose an accurate structured query-database linking.
3. Decomposition Strategies: When dealing with complex queries, it is challenging to devise effective strategies for decomposing the complicated linked structure for step-by-step SQL generation.

In this paper, we enhance the existing framework of text-to-SQL by introducing the tailored **Structure Guided SQL (SGU-SQL)** generation model. This model represents user queries and databases in a structured manner and employs a tailored semantic-enhanced structure matching model to establish a connection between the query structure and the database structure. Subsequently, we decompose the linked structure into sub-grammar trees to guide the step-by-step generation of SQLs by the LLM. The main contributions of our research are summarized as follows:

- We discover the limitation of neglecting the structure in the SQL generation task and propose our tailored Structure Guide to utilize the structure information to enhance the SQL generation.
- SGU-SQL proposes graph-based structure construction to comprehend user query and database understanding and then link query structure and database structure with semantic-enhanced solutions.
- SGU-SQL introduces tailored structure-decomposed generation strategies to decompose queries with grammar trees and then generate accurate SQL with LLM.
- Extensive experiments on two benchmark datasets verify SGU-SQL outperforms sixteen state-of-the-art SQL-generation baselines.

2 Related Work

Text-to-SQL facilitates user interaction with databases via natural language queries, circumventing the necessity for SQL syntax and database structure proficiency. Historically, this challenge was tackled through rule-based methods and handcrafted heuristics. However, these approaches often faltered due to the complexity and diversity inherent in natural language queries.

2.1 Fine-tuning Large Language Models for Text-to-SQL

The emergence of large language models (LLMs) has markedly altered the landscape for text-to-SQL tasks. LLMs, with their capacity for understanding and generating human-like text, present a robust solution for text-to-SQL applications (Liu et al., 2023). The development of LLMs typically encompasses pre-training followed by fine-tuning. Research has concentrated on fine-tuning with domain-specific data and optimization techniques to enhance base models for coding tasks, including text-to-SQL. This process enables models to master programming language syntax and database schema intricacies (Raffel et al., 2020; Roziere et al., 2023). Through training on tailored datasets of annotated SQL queries, LLMs acquire the grammar and structure necessary for generating compliant SQL code (Trummer, 2022; Sun et al., 2023). Furthermore, Scholak et al. (2021) introduced a decoding mechanism for LLMs that ensures the generation of valid sequences by discarding inadmissible tokens at each step, employing incremental parsing to guarantee the validity of SQL queries produced by auto-regressive language models.

2.2 In-Context Learning for Text-to-SQL

In-context learning enhances LLM performance by providing contextual examples during inference, thereby improving performance for specific tasks. This approach has seen innovative applications in text-to-SQL, with strategies aimed at optimizing example selection based on user queries and referenced data. Liu et al. (2021) first investigated the impact of in-context examples on GPT-3's performance. Nan et al. (2023) proposed categorizing problems by complexity to facilitate the identification of similarly complex examples, whereas Guo et al. (2023) developed a domain-specific vocabulary masking technique for example similarity assessment, highlighting the relevance of SQL-specific terms. DIN-SQL represents a sophisticated application of this concept, dissecting the text-to-SQL task into schema linking, query

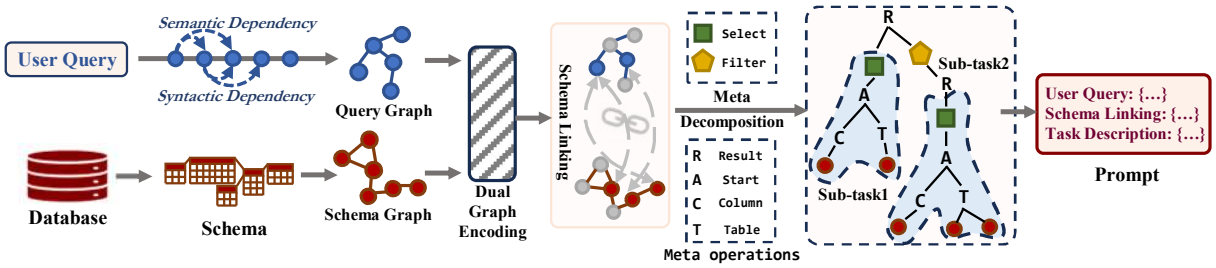

Figure 1: The overall framework.

classification, and decomposition to improve the generation of complex SQL queries (Pourreza and Rafiei, 2023). DAIL-SQL further enhanced the performance by incorporating suitable formatting of the database schema and selecting examples based on skeleton similarities(Gao et al., 2023). Dong et al. (2023b) designed a framework for ChatGPT with clear prompting for effective input layout and tailored hints for calibration and consistency check during the query generation.

2.3 Structure Learning for Text-to-SQL

As text-to-SQL models often employ an encoder-decoder architecture, many strategies such as Graph Neural Networks (GNNs) have been designed to learn structural representations for the decoders to comprehend questions and database schemas. RATSQL employs a graph-based structure to delineate relationships within database schemas and queries, treating the schema as a graph of tables and columns connected by relational edges (Wang et al., 2019). Cao et al. (2021) introduced LGESQL, an edge-centric graph model derived from conventional node-centric graphs, to capture diverse structural topologies. S2SQL, proposed by Hui et al. (2022), integrates syntactic dependency information into a question-schema interaction graph, focusing on primary relationships to mitigate overfitting while emphasizing essential graph structures. Li et al. (2023b) explored the integration of GNN layers into the T5 model, aiming to leverage both semantic and structural information from PLMs and GNNs, respectively. Li et al. (2023a) designed a ranking-enhanced encoder to rank and filter the schema items for the skeleton-aware decoder to decouple the schema linking and the skeleton parsing stage.

3 Methodology

In this section, we introduce the rationales of our framework. We leverage the implicit structural information in both queries and databases from three aspects: (i) A graph-based structure construction for both user query and database understanding; (ii) A tailored structure linking method is proposed to bind the query nodes with corresponding schema nodes, i.e., tables and columns. (iii) Structure prompting to LLMs for accurate SQL generation. The overall illustration is presented in Figure 1.

3.1 Structure Construction

Initiating LLMs with the comprehension of the user query and database schema is challenging in complex real-world scenarios. First, natural language queries often contain ambiguous references or terminologies. Second, the implicit structure information is not readily accessible in the plain text format. In this part, we introduce the details about structure extraction in both query and database to achieve effective mapping and interaction.

Inspired by the superiority of graphs, which can effectively represent the structural information in both natural language sentences and real-world entities (Dong et al., 2023a), we aim to construct build a comprehensive query-schema graph by combining the *query structure* and *databse structure* based on the topic concepts in the query, predefined relations in the query, database schema as well as the table/column names present within the schema. The graph contains three key structures:

- **Query Structure** (R_{query}): Encodes syntactic dependencies between tokens in the question, derived from its syntactic parse.
- **Database Structure** (R_{schema}): Represents intrinsic relationships within the database schema, like foreign keys.
- **Linking Structure** (R_{linking}): Aligns query entities with corresponding schema columns or tables.

To make it clear, we introduce the implementation details of query and database structure construction

in Section 6.1 of the Appendix, and propose a novel structure learning framework to obtain the linking structure in the following section.

3.2 Structure Linking

Schema linking plays a vital role in accurately interpreting and generating SQL queries from textual input.Existing schema linking has been actively adopted as a rule-based mapping problem. However, the hard mapping based on concept and table/column names results in an unsatisfactory performance. To tackle this, given the constructed query and database structures, we value the adjacency information during the matching process and propose to automatically build the connection between the query structure and schema at the node level. Specifically, we design a tailored structure-based linking framework. Both query and schema structures are first encoded through a Relational Graph Attention Transformer (RGAT) for initial node representations. The representation learning process is guided by the message propagation within the self-structure. We formalize the procedure of structure-aware question-schema structure linking as follows:

$$\mathcal{G}_s^{Agg} = \mathrm{Agg}(\mathcal{G}_s, \mathcal{G}_q), \quad (1)$$

$$\mathcal{G}_q^{Agg} = \mathrm{Agg}(\mathcal{G}_q, \mathcal{G}_s), \quad (2)$$

where the structure-aware aggregation function $\mathrm{Agg}(.)$ is employed to gather information from both the schema-graph $\mathcal{G}_s$ and the query-graph $\mathcal{G}_q$ and transfer it to the adjacent graph.

Let $\{\boldsymbol{h}_i^q\}_{i=1}^m$ represent a set of node embeddings in the query-graph $\mathcal{G}_q$ and let $\{\boldsymbol{h}_j^k\}_{j=1}^n$ denote a set of node embeddings in the subgraph $\mathcal{G}_k$ that extracted from schema-graph $\mathcal{G}_s$. In particular, we first employ global-average pooling on the node embedding $\boldsymbol{h}_i^q$ of the query structure $\mathcal{G}_q$ in order to derive the global query structure embedding $\boldsymbol{h}_g^q$. Following this, to encapsulate globally pertinent information, the key node embedding $\boldsymbol{h}_j^k$ is updated in the subsequent manner:

$$\boldsymbol{h}_g^q = \frac{1}{m}\sum\nolimits_{i=1}^m \boldsymbol{h}_i^q, \quad (3)$$

$$\alpha_j = \theta\left(\boldsymbol{h}_g^{qT}\boldsymbol{W}_g\boldsymbol{h}_j^k\right), \quad (4)$$

$$\boldsymbol{h}_j^k = \sum\nolimits_{l\in\mathcal{N}_j} \alpha_l \boldsymbol{W}_k\boldsymbol{h}_l^k + \alpha_j\boldsymbol{W}_k\boldsymbol{h}_j^k \quad (5)$$

$$+(1-\alpha_j)\boldsymbol{W}_q\boldsymbol{h}_g^q \quad (6)$$

where $\boldsymbol{W}_g$, $\boldsymbol{W}_q$, $\boldsymbol{W}_k$ represent trainable parameters, and θ illustrates a sigmoid function. While α_j denotes the relevance score situated between the j-th key node and the global query structure.

For each node a in the query structure $\mathcal{G}_q$, it is necessary to find a corresponding matching node s in the database $\mathcal{G}_s$. The proposed solution mainly consists of three steps. First, a set of most relevant candidate nodes $\{s_1, s_2, \ldots, s_K\}$ is identified through string matching in the set of tables and columns A. Second, for each candidate node s, a specialized subgraph $\mathcal{G}(a, s)$ is constructed. As shown in Figure 1, $\mathcal{G}(a, s)$ includes the query graph $\mathcal{G}_q$, adjacent nodes of s_k (first-order and second-order), and an edge connecting a and s_k. Lastly, we train a structure learning model $\mathrm{RGAT}(\cdot)$ to learn the graph-level embedding representation of $\mathcal{G}_(a, s_k)$ for predicting a matching score of the candidate pair (a, s_k) . The expression for this is as follows,

$$\boldsymbol{h} = \mathrm{RGAT}(\mathcal{G}(a, s)), \quad (7)$$

$$\mathrm{Score}_{(a,s_k)} = \sigma(\sum\nolimits_{l\in\mathcal{G}(a,s_k)} \boldsymbol{h}_l^k). \quad (8)$$

Based on positive samples (a, s) and negative samples (a, s_k), where $s_k \neq s$, the structure learning model $\mathrm{RGAT}(\cdot)$ is iteratively trained.

3.3 Structure-Decomposed Prompting

Decomposing query with grammar tree The performance of LLMs on complex tasks can be improved by using decomposing-based methods. However, decomposing a SQL query into subtasks is challenging due to its declarative structure and the intricate connections between query concepts. To this end, in this section, we introduce a context-free grammar tree (defined in Figure 2-(a)) to break the query into subtasks according to the syntax structure of the user query. Specifically, we first employ the Stanford Parser (Klein and Manning, 2003) to build the grammar tree to achieve a linguistic understanding of the natural language query and then adopt a node mapper to match nodes in the linguistic parse tree to SQL operations (Kate, 2008). Following this, the original query can be divided into several subtasks according to the SQL operations distributed on the grammar tree. More details are presented in Section 6.3 of Appendix.

Meta-operation in SQL Based on the commonly used operations and properties in SQL, we define four meta-operations as the major identifiers for sub-task division:

- **A**: a start of a `SELECT` statement;

Figure 2: The example of grammar tree. The subfigure (a) denotes the context-free grammar of SQL. While subfigures (b) and (c) demonstrate two examples of the constructed grammar tree used in our SGU-SQL.

- **R**: a root node or continue with the result returned by last SELECT;
- **T**: a corresponding Table name;
- **C**: a corresponding Column name.

Following this, the original query can be divided into several subtasks according to the SQL operations distributed on the grammar tree.

Prompt design for Text-to-SQL Given LLMs are sensitive to the prompts, we aim to transform the graph-structured sub-tasks into the form of text. This helps the LLMs better understand the task requirements to generate effective SQL queries. Specifically, given a user query q_j, to systematically guide the language model in generating SQL queries, we decompose the generation task into multiple steps and guide the language model to solve it step by step. The final prompts follow the format $< Q_j; G_j; < Q_{j1}; A_{j1} \ldots; Q_{jk}; A_{jk} >; I_j; A_j >$, where k represents the number of subtasks, Q_j and A_j represent the statement of each subtask and the relevant information involved in this task, respectively, G_j denotes the linked query-schema structure, and I_j is the task description.

4 Experiments

This section empirically evaluates the proposed framework, i.e., SGU-SQL, and presents its performance on two benchmark datasets. Our empirical study is motivated by the following questions:

- **Q1** How does our proposed SGU-SQL perform in comparison with the strongest baselines, including traditional finetuning-based, structure-learning-based methods, and other in-context-learning-based methods?
- **Q2** Could our proposed SGU-SQL enhance other LLMs by substituting the original framework with the structure-decomposing-based prompt?
- **Q3** Is our proposed structure prompting effective when handling queries of different difficulty levels? What types of queries are prone to errors in our model? What is the reason for the error?

4.1 Experiment Setup

Datasets We assess the performance of text-to-SQL models using two renowned datasets, Spider (Yu et al., 2019) and BIRD (Li et al., 2023c). Spider, a cross-domain text-to-SQL dataset, comprises 8659 instances in the training split and 1034 instances in the development split, spanning across 200 databases. Each instance comprises a natural language question related to a specific database and its corresponding SQL query. For evaluation purposes, we utilize the Spider-dev development split since the test split has not been released. On the other hand, BIRD (BIg Bench for large-scale Database Grounded text-to-SQL Evaluation) is another pioneering cross-domain dataset that focuses on exploring the impact of extensive database contents on text-to-SQL parsing. BIRD features over 12,751 unique question-SQL pairs, encompassing 95 large databases with a total size of 33.4 GB. It encompasses more than 37 professional domains, including blockchain, hockey, healthcare, education, and others.

Baselines To valid the effectiveness of SGU-SQL, we compare it with the state-of-art baselines according to our best knowledge. Following the taxonomy aforementioned in Section 2, we divide all baselines into three categories: (i) *Fine-tuning-based*: **T5-base** (Raffel et al., 2020),

Model	SPIDER		BIRD	
	Exec Acc	EM Acc	Exec Acc	EM Acc
Fine-tuning-based				
T5-Base (Raffel et al., 2020)	0.5791	0.5722	0.1154	0.1009
T5-Large (Raffel et al., 2020)	0.6728	0.6536	0.1970	0.1736
T5-3B (Raffel et al., 2020)	0.7440	0.7152	0.2334	0.2068
Structure-learning-based				
RAT-SQL + GAP (Wang et al., 2019)	0.7505	0.7379	0.2639	0.2431
RASAT + PICARD (Qi et al., 2022)	0.8048	0.7537	0.2870	0.2603
S^2SQL + ELECTRA (Hui et al., 2022)	0.7643	0.7285	-	-
RESDSQL-3B (Li et al., 2023a)	0.8173	0.7580	0.3031	0.2569
Graphix-3B + PICARD (Li et al., 2023b)	0.8121	0.7563	0.2984	0.2418
Incontext-learning-based				
CodeX (Chen et al., 2021)	0.7167	0.4905	0.3438	0.3019
PaLM-2 (Anil et al., 2023)	0.6985	0.4438	0.2735	0.2543
ChatGPT (OpenAI, 2023)	0.7394	0.5327	0.3562	0.3041
GPT-4 (OpenAI, 2023)	0.7665	0.5892	0.4633	0.4255
CoT + ChatGPT (Wei et al., 2022)	0.7552	0.5684	0.3665	0.3127
C3-GPT (Dong et al., 2023b)	0.8185	0.7036	-	-
DIN-SQL + GPT-4 (Pourreza and Rafiei, 2023)	0.8237	0.7187	0.5079	0.4398
DAIL-SQL + GPT-4 (Gao et al., 2023)	0.8449	0.7443	0.5434	0.4581
SGU-SQL + GPT-4 (Ours)	**0.8791**	**0.7679**	**0.5771**	**0.4989**

Table 1: The Execution Accuracy and Exact Match Accuracy of text-to-SQL models on SPIDER and BIRD.

T5-large (Raffel et al., 2020), **T5-3B** (Raffel et al., 2020); (ii) *structure-learning-based*: **RAT-SQL** (Wang et al., 2019), **RASAT** (Qi et al., 2022), **S^2SQL** (Hui et al., 2022) ,**RESDSQL** (Li et al., 2023a),**GRAPHIX** (Li et al., 2023b); and (iii) *incontext-learning-based*: **PaLM-2** (Anil et al., 2023), **CodeX** (Chen et al., 2021), **ChatGPT** (OpenAI, 2023), **GPT-4** (OpenAI, 2023), **C3-GPT** (Dong et al., 2023b), **DIN-SQL** (Pourreza and Rafiei, 2023), **DAIL-SQL** (Gao et al., 2023).

Evaluation Metrics The effectiveness of our models is analyzed by employing two frequently used measurement standards, namely Exact-Set-Match Accuracy (EM Acc) and Execution Accuracy (Exec Acc). EM Acc approaches treat each clause as an individual set and examine its prediction to the equivalent validated SQL query. However, this parameter does not take into consideration values, potentially triggering false positives and negatives. Conversely, Exec Acc places the execution result of the prospective SQL query alongside that of the confirmed SQL query on certain database iterations. With Execution Accuracy, a more nuanced understanding of the model's functionality is attained since there can be several legitimate SQL queries corresponding to one question, and Exact-Set-Match Accuracy only assesses the predicted SQL in contrast to one of them.

Execution accuracy					
Model	**Easy**	**Medium**	**Hard**	**Extra**	**All**
SGU-SQL + GPT-4	93.5	91.9	845	72.3	87.9
DAIL-SQL + GPT-4	90.6	82.7	75.4	58.2	84.5
DIN-SQL + GPT-4	91.5	78.4	69.4	54.3	82.4
GPT-4	87.3	76.0	63.2	51.9	76.7
CodeX	82.8	67.9	57.7	48.5	73.9
Exact match accuracy					
Model	**Easy**	**Medium**	**Hard**	**Extra**	**All**
SGU-SQL + GPT-4	85.5	83.9	72.69	51.2	76.8
DAIL-SQL + GPT-4	79.5	74.2	65.2	46.2	74.4
DIN-SQL + GPT-4	81.3	67.1	42.7	46.1	71.9
GPT-4	77.9	56.4	37.9	22.5	58.9
CodeX	65.7	42.5	28.7	15.1	49.1

Table 2: Performance of SGU-SQL and state-of-the-art in-context learning competitors on the development set of Spider with queries of different difficulty levels.

4.2 Main Results: Q1

To answer Q1, we perform extensive experiments to compare SGU-SQL with the strongest baselines

best to our knowledge. The comparison results on Spider and Bird are placed in Table 1. We summarize our experimental observations as follows.

Obs.1. Our proposed SGU-SQL significantly outperforms SOTA text-to-SQL baselines. We compare SGU-SQL (ours) with 3 fine-tuning based methods, 5 structure-learning-based methods and 8 SOTA incontext-learning-based methods over two benchmark datasets. As shown in Table 1, SGU-SQL achieves superior performance across both datasets, outperforming all baselines in terms of execution accuracy and exact match accuracy.

Obs.2. In-context learning-based method is better than the methods of the other two categories. Among the three categories of methods, in-context learning-based methods consistently demonstrate superior performance. This suggests that leveraging in-context learning mechanisms is crucial for enhancing the understanding and generation of SQL queries from natural language inputs. Specifically, the in-context learning-based methods, i.e., DIN-SQL and DAIL-SQL in our comparison set achieve higher accuracy rates and require less computational overhead compared to fine-tuning and structure-learning-based methods. Additionally, the in-context learning-based methods exhibit better generalization across different datasets, indicating their robustness and adaptability.

4.3 Ablation Staudy: Q2

The effect of prompting strategy In this part, we conduct comprehensive experiments to investigate the effectiveness of our proposed prompting strategy. Specifically, we compare the structure-based decomposing strategy used in our SGU-SQL with other prompting strategies like CoT (Wei et al., 2022) and few-shot prompting. As shown in Table 3, we can have the following observations.

Obs.1. Our structure-based decomposing significantly outperforms other prompting strategies. Our method demonstrates superior performance across all tested LLMs. Specifically, compared to CoT, our approach achieves an average improvement of 5.03%, while outperforming few-shot prompting by 4.98% on average.

Obs.2. Traditional decomposing-based methods are ineffective in the text-to-SQL task. While decomposing complex tasks into subtasks like CoT, can enhance model performance in many natural language understanding tasks, it proves to be ineffective in the text-to-SQL task. As shown in Table 3, applying COT on PaLM-2 even leads to

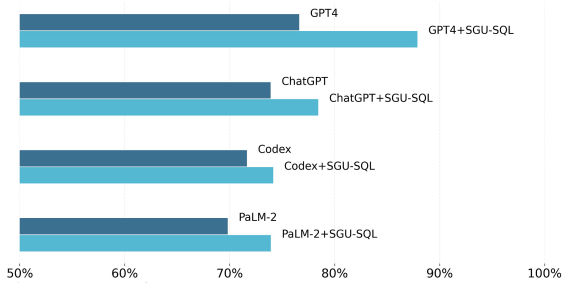

Figure 3: Performance - Exec acc on SPIDER Dev Set.

a performance decrease of 1.08% compared to the naive few-shot prompting. This is attributed to the complex syntax of SQL, and the intricate correspondence between query terms in user queries and database data units. Conversely, we formally define the meta-operations in SQL and propose a decomposing strategy according to the grammar tree to separate the query into subtasks. This boosts the LLMs' comprehension of linked queries to generate accurate SQLs step by step.

The generalization ability of prompts To further verify the generalization ability of our proposed prompting strategy, in this part, we conduct comprehensive experiments to investigate whether SGU-SQL could enhance other LLMs by substituting their original framework with the decomposing-based prompts. Specifically, we replace Gpt-4 used in SGU-SQL with other representative generative LLMs, including PaLM-2 (Anil et al., 2023), CodeX (Chen et al., 2021), ChatGPT and GPT-4 (OpenAI, 2023) as alternatives. Specifically, we used the model 'chat-bison-001' provided by GoogleAI as the implementation of PaLM-2, and 'gpt-3.5-turbo' and 'gpt-4' as the implementations of ChatGPT and GPT-4, respectively. The text-to-SQL task is conducted under the few-shot setting with the query from the development set of Spider as input. As shown in Figure 3, we have the following observations.

Obs.1. The performances of the original LLMs improved significantly by integrating the prompt learned from our SGU-SQL. Specifically, PaLM-2 improved by 4%, CodeX by 3%, ChatGPT by 5%, and GPT-4 by almost 11%. The substantial performance gains indicate the robustness and generalization ability of our proposed prompting strategy. Furthermore, the consistent improvements across different LLMs highlight the versatility and applicability of our approach in enhancing the capabilities of existing language models.

Obs.2. LLMs with stronger reasoning abilities ex-

LLM	Prompting strategy	Exec acc
PaLM-2	+ Few-shot Prompting	0.6985
	+ CoT Prompting	0.6873
	+ SGU-SQL	0.7395
CodeX	+ Few-shot Prompting	0.7167
	+ CoT Prompting	0.7198
	+ SGU-SQL	0.7418
ChatGPT	+ Few-shot Prompting	0.73947
	+ CoT Prompting	0.7552
	+ SGU-SQL	0.7846
GPT-4	+ Few-shot Prompting	0.7665
	+ CoT Prompting	0.7834
	+ SGU-SQL	0.8791

Table 3: Performance of LLMs with different prompting strategies on the development set of SPIDER.

hibit greater improvement. We observe that LLMs with stronger reasoning abilities benefit more from integrating the prompts learned from SGU-SQL. Specifically, GPT-4, which is known for its advanced reasoning capabilities, shows a more substantial performance improvement compared to PaLM-2, CodeX, and ChatGPT. This suggests that our prompting strategy is particularly effective in enhancing the performance of LLMs that require more complex reasoning tasks.

4.4 Model Analysis: Q3

Difficulty analysis In this part, we first analyze the performance of our proposed method on queries with different levels of difficulty. Our analysis focused on evaluating the performance of our proposed method across queries of varying difficulty levels. Table 2 provides a comparative assessment of our method against state-of-the-art (SOTA) prompting methods on the Spider development set. Our findings reveal that our method consistently outperforms competing methods across all difficulty levels. Notably, we observe the most substantial improvements in the extra hard and hard classes, where other prompting models struggle. Additionally, our method also shows slight improvement in the easy class, which suggests that our method is robust and effective across queries of different difficulty levels, highlighting its potential for practical applications in natural language understanding and query generation tasks.

Error analysis We checked the errors in the generated SQL answers and classified them into six categories, as shown in Figure 4 following the classification by Pourreza and Rafiei (2023). We discuss the failure cases of our model in comparison with baseline models, and then discuss the reasons for the typical failure for LLMs in Text-to-SQL tasks. Compared to the baseline model, we achieved a reduction of approximately 33.5% in errors and made progress in the schema-linking and join statement components where traditional models often falter. Please refer to Section 8 of Appendix for details.

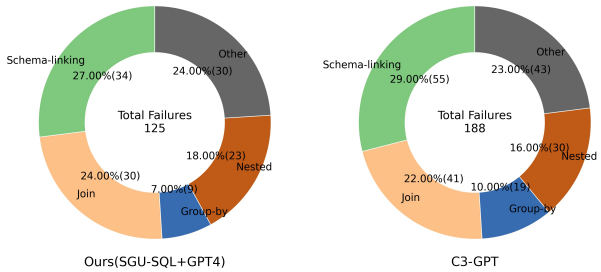

Figure 4: Error Analysis of GPT4+SGU-SQL and C3-GPT on the Dev Set: A Comparison of 125 and 188 Failures Respectively.

5 Conclusion and Future Work

Generating accurate Structured Query Language (SQL) has long been a challenging task, especially in matching users' semantic queries with structured databases. Existing models typically input queries and database schemas into LLMs to perform semantic-structure matching and generate structured SQL. However, these models often overlook the structural information inherent in user queries and databases, which could significantly enhance the generation of accurate SQL queries. This oversight can result in the production of inaccurate or unexecutable SQL queries. To fully exploit the structure, we propose the structure-to-SQL framework, which leverages the inherent structure information to improve the SQL generation of LLMs. Specifically, we introduce our **Structure Guided SQL (SGU-SQL)** generation model. SGU-SQL links user queries and databases in a structure-enhanced manner. It then decomposes complicated linked structures with grammar trees to guide the LLM to generate the SQL step by step. Extensive experiments conducted on two benchmark datasets demonstrate SGU-SQL outperforms all state-of-the-art baselines.

Limitations

The limitation of our approach is a common constraint for text-to-SQL models, as these models depend on GPT4 to generate the SQL queries. Consequently, if the network is inaccessible or the OpenAI API is blocked, the text-to-SQL models are unable to generate SQL queries as expected.

Ethics Statement

We confirm that we have fully complied with the ACL Ethics Policy in this study. We conduct experiments with widely adopted publicly available datasets. The codes and generated prompts will be open-sourced for other researchers' fair reproduction and further study in the active text-to-SQL community.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations. *arXiv preprint arXiv:2106.01093*.

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

Junnan Dong, Qinggang Zhang, Xiao Huang, Keyu Duan, Qiaoyu Tan, and Zhimeng Jiang. 2023a. Hierarchy-aware multi-hop question answering over knowledge graphs. In *Proceedings of the ACM Web Conference 2023*, pages 2519–2527.

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

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. 2023. A case-based reasoning framework for adaptive prompting in cross-domain text-to-sql. *arXiv preprint arXiv:2304.13301*.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, and Yongbin Li. 2022. S^2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers.

Rohit Kate. 2008. Transforming meaning representation grammars to improve semantic parsing. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 33–40.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*, pages 423–430.

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

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. *arXiv preprint arXiv:2301.07507*.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023c. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. Demonstration of insightpilot: An llm-empowered automated data exploration system. *arXiv preprint arXiv:2304.00477*.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies. *arXiv preprint arXiv:2305.12586*.

OpenAI. 2023. Gpt-4 technical report.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*.

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

Immanuel Trummer. 2022. Codexdb: Synthesizing code for query processing from natural language instructions using gpt-3 codex. *Proceedings of the VLDB Endowment*, 15(11):2921–2928.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task.

6 The Framework of SGU-SQL

6.1 Structure Construction

Initiating LLMs with the comprehension of the user query and database schema is challenging in complex real-world scenarios. First, natural language queries often contain ambiguous references or terminologies. Second, the implicit structure information is not readily accessible in the plain text format. In this part, we introduce the details about structure extraction in both query and database to achieve effective mapping and interaction.

Inspired by the superiority of graphs, which can effectively represent the structural information in both natural language sentences and real-world entities (Dong et al., 2023a), we aim to construct the *query structure* and *schema structure* based on the topic concepts in the query, pre-defined relations in the query, database schema as well as the table/-column names present within the schema.

Query Structure Traditional methods mainly relied on linking structures and schema structures, often overlooking the query's structure. In this paper, we integrate both semantic dependency and syntactic dependency to model the query structure, which could help LLMs accurately capture user intentions. Specifically, given a user query q_i in natural language, the corresponding query structure $\mathcal{G}_q$ is denoted hereunder,

$$\mathcal{G}_q = (\mathcal{A}, \mathcal{E}_s \cup \mathcal{E}_o), \tag{9}$$

where $\mathcal{A}$ represents the set of keywords in the query. $\mathcal{E} = \{\mathcal{E}_s \cup \mathcal{E}_o\}$ defines the syntactic and order dependencies among these keywords. The dependency $\mathcal{E}_s$ signifies the semantic-level relationship, while $\mathcal{E}_o = \{e_1, e_2\}$ respectively represent the first-order and second-order adjacency relationships in terms of the structural arrangement. For further demonstration and examples, we introduce the implementation details in the Appendix.

The query structure provides a novel perspective of structured understanding on natural language sentences. This provides finely divided requirements for the model to understand exact intentions based on the dependencies between keywords.

Database Structure Since database schema is already a highly condensed structural summary, we leverage this information to construct a database structure by introducing relations within and across tables. Specifically, each table and column in the database is represented as a table/column node in the database structure, denoted as $\mathcal{S}$. According to the nature of the database, they are connected by three types of pre-defined relations, denoted as $\mathcal{E}_d = \{e_t, e_p, e_f\}$. e_r represents the association relationship between a table and the contained columns. e_p and e_f are for the primary key and foreign key relationship, respectively. In general, the schema structure $\mathcal{G}_s$ is extracted as:

$$\mathcal{G}_s = (\mathcal{S}, \mathcal{E}_d). \tag{10}$$

We transform the readily available database schema into a schema structure, providing a more profound and perspective of the intrinsic connections between database contents. This aligns with the query structure and is employed to perform effective mapping in the following section.

6.2 Structure Linking

To successfully convert a natural language question into a structured query, it is crucial to establish a connection between the query concepts, i.e., words or phrases in the question and the corresponding database contents, i.e., tables and columns in the database. This process, known as schema linking, plays a vital role in accurately interpreting and generating SQL queries from textual input. In this paper, we propose a structure-based linking framework that matches user queries and databases more correctly. After we get the accurate linking from the structure learning model, we further incorporated three additional relations to supplement effective connections between user query and database schema, which is defined in Table 4.

6.3 Prompt Design for Text-to-SQL

6.3.1 Structure-based decomposing

Decomposing a SQL query into subtasks is challenging due to its declarative structure and the intricate connections between query concepts. To this end, we introduce a context-free grammar tree to break the query into subtasks according to the syntax structure of the user query. Specifically, we first employ the Stanford Parser (Klein and Manning, 2003) to build the parsing tree to achieve a linguistic understanding of the natural language query, and then adopt a node mapper to match nodes in the linguistic parse tree to SQL operations following previous study (Kate, 2008).

6.3.2 Meta-operation in SQL

Based on the commonly used operations and properties in SQL, we define four meta-operations as

Strucure	Source x	Target y	Type	Description
Question Structure	Question	Question	Forward-Syntax	y is the target word of x under syntax dependency.
	Question	Question	Backward-Syntax	y is the source word of x under syntax dependency.
	Question	Question	None-Syntax	x and y have no syntactic dependency.
Query Structure	Column	Column	Foreign-Key	y is the foreign key of x.
	Table	Column	Has	The column y belongs to the table x.
	Table	Column	Primary-Key	The column y is the primary key of the table x.
Linking Structure	Question	Table	None-Linking	No linking between x and y.
	Question	Table	Partial-Linking	x is part of y, but the entire question does not contain y.
	Question	Table	Exact-Linking	x and y are matched based on our Structure Linking model.
	Question	Column	None-Linking	No linking between x and y.
	Question	Column	Partial-Linking	x is part of y, but the entire question does not contain y.
	Question	Column	Exact-Linking	x and y are matched based on our Structure Linking model.
	Question	Column	Value-Linking	x is part of the candidate cell values of column y.

Table 4: The list of all relations used in three structures in SU-SQL. All relations above are asymmetric.

the major identifiers for sub-task division:

- **A**: a start of a SELECT statement;
- **R**: a root node or continue with the result returned by last SELECT;
- **T**: a corresponding Table name;
- **C**: a corresponding Column name.

Following this, the original query can be divided into several subtasks according to the SQL operations distributed on the grammar tree.

7 Case Study

7.1 Case 1: Airport Database Question

Here we use DALI-SQL as a baseline to generate prompts for GPT-4 to reason, where we followed experimental settings and codes provided by Gao et al. (2023) to compute similarities for example selections and prompt formatting. In particular, we use 9-shot examples with masked similarities and COMPLETE prompt format. We select some of the failure cases to demonstrate the common mistakes made by LLMs with the main-stream methods for in-context learning.

Prompts generated by baseline models

```
/* Given the following database schema: */
CREATE TABLE 'flights' (
    Airline INTEGER,
    FlightNo INTEGER,
    SourceAirport TEXT,
    DestAirport TEXT,
    PRIMARY KEY(Airline, FlightNo),
    FOREIGN KEY (SourceAirport)
        REFERENCES airports(AirportCode),
    FOREIGN KEY (DestAirport) REFERENCES airports(AirportCode)
);
/* Answer the following: What is the code of airport that has the highest number of flights? */;
Let's think step by step.
```

Returned answers from our model

```
SELECT SourceAirport, COUNT(*) AS NumberOfFlights
FROM flights
GROUP BY SourceAirport
ORDER BY NumberOfFlights DESC
LIMIT 1;
```

Gold Query

```
SELECT T1.AirportCode
FROM AIRPORTS AS T1
JOIN FLIGHTS AS T2 ON T1.AirportCode = T2.DestAirport
   OR T1.AirportCode = T2.SourceAirport
GROUP BY T1.AirportCode
ORDER BY count(*) DESC
LIMIT 1;
```

7.2 Case 2: Languages Database Question

Prompts generated by baseline models

```
/* Given the following database schema: */
...

CREATE TABLE ‘countrylanguage‘ (
  ‘CountryCode‘ char(3) NOT NULL DEFAULT ''
  , ‘Language‘ char(30) NOT NULL DEFAULT ''
  , ‘IsOfficial‘ text  NOT NULL DEFAULT 'F'
  , ‘Percentage‘ float(4,1) NOT NULL DEFAULT '0.0'
  , PRIMARY KEY (‘CountryCode‘,‘Language‘)
  , CONSTRAINT ‘countryLanguage_ibfk_1‘ FOREIGN KEY (‘CountryCode‘) REFERENCES ‘country‘ (‘Code‘)
)
```

Figure 5: The example of grammar tree. The subfigure (a) denotes the context-free grammar of SQL. While subfigures (b) and (c) demonstrate two examples of the constructed grammar tree.

```
/* Answer the following: How many people live in countries that do not speak English? */
Let's think step by step.
```

Returned answers from our model

```
SELECT SUM(c.Population)
FROM country c
WHERE c.Code NOT IN (
    SELECT cl.CountryCode
    FROM countrylanguage cl
    WHERE cl.Language = 'English' AND cl.IsOfficial = 'T'
);
```

Gold Query

```
SELECT sum(Population)
FROM country
WHERE Name NOT IN (
    SELECT T1.Name
    FROM country AS T1
    JOIN countrylanguage AS T2
    ON T1.Code = T2.CountryCode
    WHERE T2.Language = "English"
);
```

7.3 Case 3: Students Database Question

Prompts generated by baseline models

```
/* Given the following database schema: */
...

CREATE TABLE ‘Transcript_Contents‘ (
‘student_course_id‘ INTEGER NOT NULL,
‘transcript_id‘ INTEGER NOT NULL,
FOREIGN KEY (‘student_course_id‘ )
    REFERENCES ‘Student_Enrolment_Courses‘(‘student_course_id‘ ),
FOREIGN KEY (‘transcript_id‘ )
    REFERENCES ‘Transcripts‘(‘transcript_id‘ )
)
```

```
/* Answer the following: What is the id of the semester that had both Masters and Bachelors students enrolled? */
Let's think step by step.
```

Returned answers from our model

```
SELECT SE.semester_id
FROM Student_Enrolment AS SE
JOIN Degree_Programs AS DP ON SE.degree_program_id = DP.degree_program_id
WHERE DP.degree_summary_name IN ('Masters', 'Bachelors')
GROUP BY SE.semester_id
HAVING COUNT(DISTINCT DP.degree_summary_name) > 1;
```

Gold Query

```
SELECT DISTINCT T2.semester_id FROM Degree_Programs AS T1 JOIN Student_Enrolment AS T2 ON T1.degree_program_id = T2.degree_program_id WHERE degree_summary_name = 'Master' INTERSECT SELECT DISTINCT T2.semester_id FROM Degree_Programs AS T1 JOIN Student_Enrolment AS T2 ON T1.degree_program_id = T2.degree_program_id WHERE degree_summary_name = 'Bachelor';
```

7.4 Case 4: Social Network Database Question

Prompts generated by baseline models

```
/* Given the following database schema: */
...

CREATE TABLE Likes(
        student_id int,
        liked_id int,
        primary key (student_id, liked_id),
```

```
        foreign key (liked_id)
            references Highschooler(ID),
        foreign key (student_id)
            references Highschooler(ID)
)

/* Answer the following: Return the
    number of likes that the high
    schooler named Kyle has. */
Let's think step by step.
```

Returned answers from our model

```
SELECT count(*)
FROM Likes
JOIN Highschooler
ON Likes.liked_id = Highschooler.ID
WHERE Highschooler.name = 'Kyle';
```

Gold Query

```
SELECT count(*) FROM Likes AS T1 JOIN
    Highschooler AS T2 ON T1.student_id
     =  T2.id WHERE T2.name  =  "Kyle";
```

8 Failure Analysis

In this section, we will first discuss the failure cases of our model in comparison with baseline models, and then discuss the reasons for the typical failure for LLMs in text-sql tasks. We checked the errors in the generated SQL answers and classified them into six categories, as shown in Figure 4 followed by the major classification by Pourreza and Rafiei (2023).

Compared to the baseline model, we achieved a reduction of approximately 33.5% in errors and made progress in the schema-linking and join statement components where traditional models often falter. Errors in the schema-linking segment decreased by around 38%, primarily attributed to the utilization of Precise Query Matching, wherein graph neural networks were employed to learn and match the database schema. This underscores the efficacy of Structure Linking. In the sections prone to errors, such as Group-by and Join, our errors decreased by 35%, indicating that our grammar tree decomposing enables the model to more accurately utilize corresponding SQL Meta-operations to achieve the intention queries, thus further enhancing the accuracy in identifying the targeted tables or columns for manipulation.

To look in detail, the reason why the query provided by GPT-4 with prompts from the baseline model can be divided into several aspects.

Subtask Decomposing This is the situation where the LLMs often do not adequately break down the task into its essential steps for reasoning. For example, in Case 1, the primary subtask of linking flight data to specific cities was ignored. The question did not adequately break down the task into its essential components without further guidance by LLMs. In Case 3, the query did not decompose the task into two separate subtasks to identify semesters with Masters and Bachelors enrollments independently which also leads to wrong returned answers.

Intention Understanding LLMs sometimes misunderstand the core intention of the question. In Case 2, LLMs fail to identify the intention that the question is trying to find all countries where English is spoken, regardless of its official status which leads to errors. It concentrated on the official language status, which did not align with the broader objective of considering English-speaking countries in general. In Case 1, the query was centered around airport codes (SourceAirport), misinterpreting the intention to identify the busiest city, not just the airport. In Case 4, LLM misinterprets the intention of finding how many likes Kyle has received. It erroneously assumes the task is to count how many likes Kyle has given, not received.

Data Schema Linking Since LLMs get data schema information with plain text as inputs, it might be challenging to reason the right linking strategy to solve the problem correctly. It needs to understand the referenced tables and columns in the question which are often being mentioned in an inexplicitly way, then matching with the database schema. In contrast, our tailored GNN model can handle this situation well. In Case 1, the initial query failed to incorporate the airport's table, which was essential for linking airport codes to their respective cities. In Case 3, the query did not effectively link degree program types (Masters, Bachelors) to semesters in databases in a way that would allow for the inclusive identification of valid semesters. In Case 4, there was a misalignment in linking: the `student_id` from the `Likes` table was incorrectly associated with the `id` in the `Highschooler` table. The correct approach should link `liked_id` from `Likes` to `id` in `Highschooler` to align with the task's objective.