KeyInst: Keyword Instruction for Improving SQL Formulation in Text-to-SQL

Anonymous ACL submission

Abstract

Text-to-SQL parsing involves the translation of natural language queries (NLQs) into their corresponding SQL commands. A principal challenge within this domain is the formulation of SQL queries that are not only syntactically correct but also semantically aligned with the natural language input. However, the intrinsic disparity between the NLQ and the SQL poses a significant challenge. In this research, we introduce *Keyword Instruction (KeyInst)*, a novel method designed to enhance SQL formulation by Large Language Models (LLMs). KeyInst essentially provides guidance on pivotal SQL keywords likely to be part of the final query, thus facilitates a smoother SQL query formulation process. We explore two strategies for integrating KeyInst into Text-to-SQL parsing: a pipeline strategy and a single-pass strategy. The former first generates KeyInst for question, which are then used to prompt LLMs. The latter employs a fine-tuned model to concurrently generate KeyInst and SQL in one step. We developed *StrucQL*, a benchmark specifically designed for the evaluation of SQL formulation. Extensive experiments on StrucQL and other benchmarks demonstrate that KeyInst significantly improves upon the existing Text-to-SQL prompting techniques.

1 Introduction

The task of Text-to-SQL parsing, which aims at translating natural language questions into executable SQL queries, has gained increasing attention in recent years, as it can help non-expert users quickly access information in the database without the need for technical background (Deng et al., 2021; Yu et al., 2020; Rajkumar et al., 2022; Ni et al., 2023). Text-to-SQL parsing faces two main challenges: schema linking and SQL formulation. *Schema linking* involves identifying the pertinent tables and columns in a database schema in response to an NLQ. *SQL formulation* refers to generating SQL queries that are not only syntactically correct but also semantically aligned with the natural language input.

This paper primarily focuses on the challenge of SQL formulation. Currently, most Text-to-SQL prompting methods induce Large Language Models (LLMs) to generate the target SQL directly using In-context Learning (ICL) (Nan et al., 2023; Pourreza and Rafiei, 2024a; Tan et al., 2024). However, the vast difference between natural language queries (NLQ) and SQL hinders precise query formulation. In previous works, the skeleton-aware decoder (Li et al., 2023) was proposed to alleviate this challenge by initially generating an SQL skeleton followed by the full query. An SQL skeleton is a basic framework of an SQL query consisting of SQL operators, without specific details such as column names, table names, or conditions. Incorporating SQL skeleton in prompting has also proven to be effective (Gao et al., 2023; Guo et al., 2023). In this work, we also use the SQL structure as a central element in SQL formulation, with a particular emphasis on identifying key SQL operators. For instance, in translating the NLQ "List the customers' first and last names from the 10 least expensive invoices", accurately identifying `ORDER BY` and `LIMIT` is crucial for formulating the correct SQL query.

We introduce *Keyword Instruction (KeyInst)*, a novel method designed to enhance SQL formulation by LLMs. KeyInst essentially provides guidance on pivotal SQL keywords likely to be part of the final query. Recognizing that SQL queries corresponding to different NLQs require distinct keywords, KeyInst adapts dynamically to each query. An example of KeyInst in action is depicted in Figure 1A, demonstrating how it analyzes a given NLQ and deduces the critical SQL keywords. This strategy effectively narrows the gap between NLQ and SQL, facilitating a smoother SQL query formulation process.

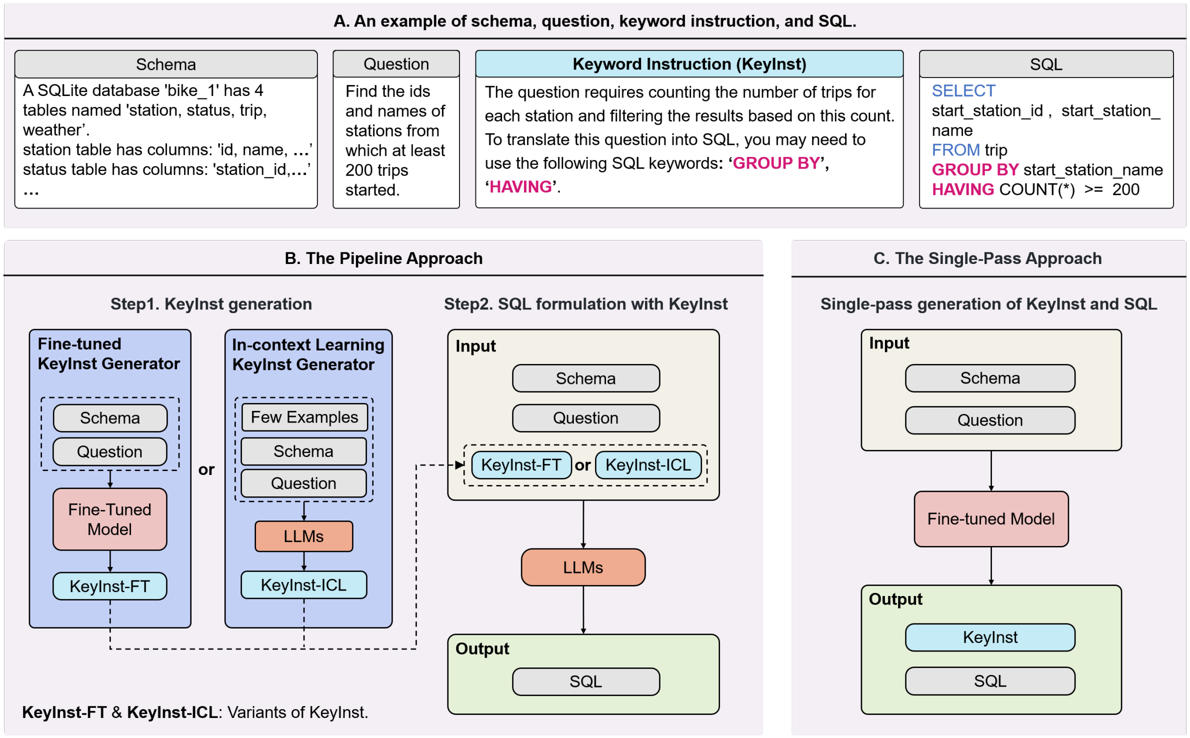

Figure 1: Graphical illustration of KeyInst and its applications: A. An example of schema, question, KeyInst, and SQL, B. The pipeline approach of KeyInst application, C. The single-pass approach of KeyInst application.

While KeyInst significantly aids in SQL query formulation, further exploration is needed on its generation and integration into Text-to-SQL parsing. We present two approaches for KeyInst generation: a model fine-tuning method and an ICL-based method. The former fine-tunes a model to produce KeyInsts for specific queries, while the latter prompts LLMs to generate KeyInsts through ICL (Brown et al., 2020). For the application of KeyInst in Text-to-SQL tasks, we also investigate two strategies. The first strategy prompts LLMs to produce SQL queries with KeyInst. The second strategy is a fine-tuning strategy that generates SQL queries directly following KeyInst generation, treating KeyInst creation as a preliminary reasoning step.

To amalgamate KeyInst generation and application within Text-to-SQL, we introduce a two-fold strategy. The pipeline approach initially generates KeyInst using either the fine-tuned or ICL-based method, followed by prompting LLMs with the generated KeyInst, as illustrated in Figure 1B. Conversely, the single-pass approach employs a fine-tuned model to concurrently generate KeyInst and SQL in one step, as depicted in Figure 1C.

Several benchmarks, such as Spider (Yu et al., 2018) and Bird (Li et al., 2024b), have been developed to assess Text-to-SQL systems. However, these benchmarks focus on overall parsing performance and lack mechanisms for isolating evaluations of semantic linking and SQL formulation. To specifically assess SQL formulation capabilities, a new benchmark called *StrucQL (Structural Benchmark for Text-to-SQL)* has been developed, derived from Spider. In StrucQL, questions and schemas are simplified: questions explicitly mention schema items, and irrelevant tables and columns are omitted from the schema. This simplification makes schema linking straightforward, shifting the primary challenge to SQL formulation. Consequently, StrucQL serves as an effective tool for evaluating SQL formulation proficiency in Text-to-SQL systems.

KeyInst was assessed on StrucQL and other benchmarks, with outcomes indicating that keyword instructions are a valuable intermediary for Text-to-SQL parsing, whether applied independently or in conjunction with other techniques.

The main contributions of this work are summarized as follows:

- We propose KeyInst, a keyword instruction tailored for each Text-to-SQL task, to alleviate SQL formulation challenge. We offer two approaches for integrating KeyInst into

Text-to-SQL parsing: a pipeline strategy and a single-pass strategy.

- The StrucQL benchmark was developed to specifically assess the SQL formulation abilities of Text-to-SQL systems. By simplifying questions and schemas, StrucQL eliminates schema linking challenges, focusing evaluation on SQL formulation performance.

- Comprehensive experiments across various benchmarks were conducted. The findings demonstrate that KeyInst significantly improves upon the existing state-of-the-art Text-to-SQL prompting techniques, showcasing its effectiveness and potential.

2 Methods

This paper introduces KeyInst to address the challenge of SQL formulation in Text-to-SQL parsing. The main idea is to analyze the NLQ to understand its intent, providing explicit guidance for SQL formulation by identifying essential keywords crucial for translating the NLQ into the target SQL. KeyInst is generated in real-time for each Text-to-SQL task.

We prepared over 6,200 KeyInst examples from the Spider training set, organized into a KeyInst set $S_{KeyInst} = \{(D_i, Q_i, K_i, S_i)\}$, where D_i is the database schema, Q_i is the question, S_i is the SQL, and K_i is the KeyInst.

Each KeyInst consists of two parts: question analysis and keyword suggestion, as shown in Figure 2. The question analysis is generated by prompting LLMs (see Appendix A). For keyword suggestions, we parse the SQL structure to identify all the keywords it uses, then filter out non-essential ones. Keywords are prioritized as follows: highest priority (`GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `EXCEPT`, `INTERSECT`, `UNION`, `WHERE`), second priority (`SELECT`, `FROM`). Lower priority keywords are only added if an SQL lacks higher priority keywords. Other Keywords(`JOIN`, `COUNT`, `IN`, and `others`) are excluded from the keyword suggestions. Without keyword prioritization, KeyInst would degrade into an SQL skeleton, which includes all the keywords of an SQL statement. More details about the skeleton can be found in Appendix B.

We implement the applications of KeyInst based on the KeyInst set $S_{KeyInst}$, detailed in the following sections.

Question:
Find the ids and names of stations from which at least 200 trips started.

KeyInst:

Question analysis
The question requires counting the number of trips for each station and filtering the results based on this count.

Keyword suggestion
To translate this question into SQL, you may need to use the following SQL keywords: **'GROUP BY', 'HAVING'**.

Figure 2: An example of the KeyInst.

2.1 Pipeline Approach of KeyInst

We propose the pipeline approach as one application of KeyInst, generating SQL in two steps. First, generating the tailored KeyInst for the each Text-to-SQL task, then prompt LLMs with the generated KeyInst to generate SQL query. This section details this application.

2.1.1 Fine-tuned KeyInst Generator

One approach for KeyInst generation is to fine-tune a model to become a KeyInst generator. Using supervised fine-tuning, the input is a database shcema D_i a question Q_i, and the target output is the corresponding KeyInst K_i. The primary objective is to minimize the following loss function:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\mathcal{M}_{\theta}\left(D_i, Q_i\right), K_i\right), \tag{1}$$

where $\mathcal{L}$ represents the loss related to the model's next token prediction, comparing the predicted KeyInst with the actual ground truth. This fine-tuned model, referred to as the *fine-tuned KeyInst generator*, analyzes the question and generates a tailored KeyInst, called *KeyInst-FT*, to prompt LLMs in SQL formulation.

2.1.2 In-context Learning KeyInst Generator

Another apporach for KeyInst generation is to prompt LLMs with ICL to generate KeyInst. We select a few examples (i.e., demonstrations) from the KeyInst set $S_{KeyInst}$ to form a few-shot prompt for generating a tailored KeyInst for a Text-to-SQL task. Each example contains a database schema D_i, a question Q_i and its corresponding KeyInst K_i. For each Text-to-SQL task, we select the top-m most similar examples based on masked question similarity (Gao et al., 2023) and combine them with the current database schema D and question Q to create a few-shot prompt. This few-shot prompt guides the LLMs to generate the tailored KeyInst, named *KeyInst-ICL*, for the current question. The process can be formulated as:

$$P_{\text{LLMs}}(y \mid x) = P\left(y \mid \text{prompt}\left((D, Q), \{(D_i, Q_i, K_i)\}_{i<=m}\right)\right), \tag{2}$$

where x is the LLMs' input, including the current schema and question and the m examples. The output y is the expected KeyInst-ICL for the current question. This system is referred to as the *in-context learning KeyInst generator*.

2.1.3 SQL formulation with KeyInst

SQL formulation follows KeyInst generation. A basic usage is to combine it with the database schema D, question Q, and KeyInst K to construct a zero-shot prompt that guides LLMs to generate SQL. This can be formulated as:

$$P_{\text{LLMs}}(y \mid x) = P\left(y \mid \text{prompt}\left(D, Q, K\right)\right), \tag{3}$$

where x is the LLMs' input (i.e., the zero-shot prompt), and y is the expected SQL output.

Notably, KeyInst functions as an instruction to enhance the SQL formulation capabilities of LLMs. It is highly extensible and can be effortlessly integrated with existing Text-to-SQL prompting methods. By appending KeyInst to these methods' prompts, their performance can be significantly improved. This will be analyzed in detail in §4.3.

2.2 Single-Pass Approach of KeyInst

We propose a single-pass approach for KeyInst as an alternative application method of KeyInst. In this approach, a fine-tuned model simultaneously generates both KeyInst and SQL in a single pass. The generation of KeyInst serves as an initial reasoning step, and the fine-tuning process helps the model internalize this reasoning, thereby improving its ability to formulate SQL queries.

This involves supervised fine-tuning, where inputs are the database schema D_i and the question Q_i, and targets are the KeyInst K_i and the SQL statement S_i. The objective is to minimize the empirical loss:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\mathcal{M}_{\theta}\left(D_i, Q_i\right), \left(K_i, S_i\right)\right), \tag{4}$$

where $\mathcal{L}$ represents the loss related to the model's next token prediction, comparing the predicted KeyInst and SQL with the actual ground truth.

The key difference between our fine-tuned model and a common Text-to-SQL fine-tuning model lies in the output. Our model first generates a KeyInst before generating the SQL, effectively reasoning about SQL formulation. Fine-tuning enables the model to remember this reasoning pattern, so it can spontaneously perform the reasoning when encountering a Text-to-SQL task.

3 StrucQL: A Structural Benchmark for Text-to-SQL

StrucQL is developed to allow researchers to swiftly and independently assess the SQL formulation capabilities of Text-to-SQL systems. Text-to-SQL errors can be categorized into semantic errors, which reflect schema linking capabilities, and structural errors, which pertain to SQL formulation skills.

However, widely-used Text-to-SQL benchmarks such as Spider (Yu et al., 2018) and Bird (Li et al., 2024b) focus on overall parsing performance and lack mechanisms for isolating evaluations of SQL formulation. Previous studies have relied on expensive manual evaluations to gauge structural performance (Ning et al., 2024), underscoring the necessity for a specialized structural benchmark for Text-to-SQL systems.

SQL structural errors fall into two categories: (1) syntax errors, such as mismatched parentheses, which make SQL unexecutable, and (2) structural misalignments with the NLQ, such as inappropriate keyword usage. While LLMs can easily generate syntactically correct SQL due to extensive pre-training, the real challenge is ensuring structural alignment with the NLQ. StrucQL, therefore, focuses on evaluating this alignment.

We developed StrucQL by modifying the Spider dataset and utilizing GPT4[1] for assistance. StrucQL comprises 1050 examples and covers 7 types of SQL operation. Each example is dedicated to a single operation type. To mitigate schema linking difficulties in the Text-to-SQL task, we implemented schema simplification. This process reduces schema linking errors, thereby providing a clearer assessment of SQL formulation. Specifically, we replaced schema-related terms in the original NLQs with the corresponding table and column names, and filtered out tables and columns from the database schema that were irrelevant to the questions (see Appendix C for an example).

Table 1 presents the performance of various LLMs on StrucQL in a zero-shot scenario. To as-

[1] https://openai.com

Type	Gemma-7B	Llama3-8B	Llama3-70B	Claude3	GPT4
Original question and schema (Input)					
GROUP BY	47.3	63.3	75.3	72.0	**75.3**
HAVING	50.0	68.0	80.0	83.3	**86.0**
ORDER BY	52.7	76.0	82.0	90.7	**92.7**
LIMIT	44.0	54.0	66.0	74.7	**80.0**
EXCEPT	36.7	59.3	63.3	68.7	**71.3**
INTERSECT	30.0	48.7	59.3	69.3	**70.7**
UNION	17.3	37.3	46.7	53.3	**56.7**
Overall	39.7	58.1	67.5	73.1	**76.1**
Schema-simplified question and schema (Input)					
GROUP BY	58.7	69.3	**78. 0**	73.3	76.7
HAVING	52.7	74.0	82.0	85.3	**86.0**
ORDER BY	66.7	74.7	90.7	92.7	**94.0**
LIMIT	51.3	76.0	80.7	77.3	**88.7**
EXCEPT	41.3	63.3	64.7	70.7	**72.7**
INTERSECT	42.7	57.3	61.3	**73.3**	72.7
UNION	30.7	38.7	47.3	55.3	**57.3**
Overall	49.1	63.9	72.0	75.4	**78.3**

Table 1: The results of execution accuracy for all compared models on StrucQL.

sess the impact of schema simplification on the Text-to-SQL task, we compared the results of models using original inputs with those using schema-simplified inputs. Overall, schema simplification improved execution accuracy. Larger models exhibited smaller improvements: GPT-4's accuracy increased by 2.2%, whereas Gemma-7B's accuracy rose by 9.4%. Additionally, we observed a significant performance gap for set operation types (`EXCEPT`, `INTERSECT`, `UNION`) compared to other types. This disparity may be due to the relative infrequency of set operations, leading to less representation in the models' training datasets. Moreover, schema simplification did not significantly enhance execution accuracy for these types, suggesting that their primary challenges are not related to schema linking issues.

To the best of our knowledge, StrucQL is the first effective tool designed to evaluate SQL formulation proficiency in Text-to-SQL systems. It offers deeper insights into the methodologies of these systems. By focusing on various types of SQL operations, StrucQL allows for a targeted evaluation of specific operations, helping to identify particular strengths and weaknesses in SQL formulation. Ultimately, this leads to more robust and accurate Text-to-SQL systems, enhancing database interactions.

4 Experiments

In this section, we systematically assess the effectiveness of KeyInst. Our evaluation centers on two primary aspects: (1) comparing the performance of different applications of KeyInst, and (2) examining the performance improvements when KeyInst is integrated with current state-of-the-art (SOTA) Text-to-SQL prompting methods.

4.1 Setup

Models We selected five LLMs for our experiments: Gemma-7B-It (Team et al., 2024) (Gemma-7B) , Llama-3-8B-Instruct[2] (Llama3-8B) , Llama-3-70B-Instruct[2] (Llama3-70B), Claude-3-Opus-20240229[3] (Claude3), and GPT-4-Turbo-2024-04-09 [1] (GPT4).

Hyperparameters For fine-tuning method, the Llama3-8B model is trained on Nvidia Tesla A100 GPUs, employing a batch size of 32 with a learning rate of 1*e-5. For the prompting method, we perform greedy decoding at a temperature of $\tau = 0$ to ensure reproducible results.

Benchmarks We used the following benchmarks: StrucQL, Spider (Yu et al., 2018), and Bird (Li et al., 2024b). StrucQL, introduced in this paper, evaluates the SQL formulation of Text-to-SQL systems. Spider is a large-scale benchmark with 8,000 training samples and 1,034 development samples across multiple databases. The BIRD dataset features 12,751 question-SQL pairs, covering 95 large databases across 37 professional fields.

Metrics We use execution accuracy (EX) to evaluate different methods. This metric compares the execution output of the predicted SQL query with that of the ground truth SQL query on same database instances.

Baselines We selected three SOTA Text-to-SQL prompting methods as the baselines.
(1) *DIN-SQL* (Pourreza and Rafiei, 2024a): This pipeline prompting method that involves schema linking, difficulty classification, SQL generation, and SQL self-correction.
(2) *DAIL-SQL* (Gao et al., 2023): An efficient few-shot prompting method that selects demonstrations based on similarity of masked question and SQL skeleton. We use DAIL-SQL with 8 shots.
(3) *SC-SQL* (Tan et al., 2024): This method integrates multiple Text-to-SQL reasoning paths and selects the best candidate result from these paths.

Our methods We introduce KeyInst and apply it

[2] https://github.com/meta-llama/llama3
[3] https://claude.ai

using two approaches: the pipeline approach and the single-pass approach. The specific implementations are:

(1) *KeyInst-FT*: A variant of KeyInst generated by the fine-tuned KeyInst generator (§2.1.1), used to prompt LLMs to generate SQL. Specifically, we fine-tuned a Llama3-8B model as the fine-tuned KeyInst generator.

(2) *KeyInst-ICL*: Another variant of KeyInst generated by the in-context learning KeyInst generator (§2.1.2). For each Text-to-SQL task, 6 examples are retrieved based on masked question similarity (Gao et al., 2023) to create a few-shot prompt, which is then used to guide GPT4 in generating KeyInst-ICL.

(3) *KeyLla*: As described in §2.2, we fine-tuned a Llama3-8B model with the KeyInst set, resulting in the KeyLla model. This model can perform KeyInst reasoning first and then generate SQL.

4.2 Results on StrucQL

We introduce StrucQL as a benchmark designed to evaluate the performance of Text-to-SQL systems in the challenge of SQL formulation. In this section, we analyze the performance of various applications of KeyInst on the StrucQL benchmark.

4.2.1 Comparison of KeyInst Applications

we compared two applications of KeyInst: the pipeline approach and the single-pass approach. The KeyLla, a Llama3-8B model fine-tuned with KeyInst, represents the single-pass approach. To ensure a fair comparison, we also used the Llama3-8B model within the pipeline to generate SQL. We utilized two variants of KeyInst: KeyInst-FT and KeyInst-ICL. Despite their different generation methods, both variants serve as part of the prompt to guide the Llama3-8B in SQL formulation. The results are detailed in Table 2.

Table 2 demonstrates that the single-pass approach (i.e., KeyLla) outperforms the pipeline approach (i.e., KeyInst-FT and KeyInst-ICL) in terms of performance. KeyLla effectively internalizes KeyInst's reasoning for SQL formulation, leading to superior results. While the single-pass approach achieved the best performance, the pipeline approach can also achieve comparable results. Additionally, the pipeline approach have the advantage of being able to leverage more powerful LLMs, such as GPT4. Achieving similar performance through single-pass approach would be significantly more costly.

Type	Single-Pass	Pipeline	
	KeyLla	KeyInst-FT	KeyInst-ICL
GROUP BY	**80.7**	74.5	79.3
HAVING	**85.3**	80.0	82.0
ORDER BY	**90.7**	88.0	87.3
LIMIT	83.3	82.7	**86.7**
EXCEPT	**78.0**	69.3	73.3
INTERSECT	**82.0**	81.3	80.7
UNION	53.3	**62.0**	54.0
Overall	**79.1**	76.9	76.8

Table 2: Execution accuracy (EX) of KeyInst applications on StrucQL. KeyLla is a fine-tuned Llama3-8B model. KeyInst-FT and KeyInst-ICL are variants of KeyInst, used to prompt LLMs (here, the Llama3-8B) for SQL formulation.

Models	Prompting	EX
Llama3-8B	KeyInst-ICL	76.8
	KeyInst-FT	**76.9**
Llama3-70B	KeyInst-ICL	80.5
	KeyInst-FT	**83.2**
GPT4	KeyInst-ICL	82.3
	KeyInst-FT	**84.3**

Table 3: Execution accuracy (EX) of various LLMs using KeyInst-FT and KeyInst-ICL on StrucQL.

In conclusion, both single-pass and pipeline are effective for addressing the SQL formulation challenge in Text-to-SQL tasks. If budget allows, single-pass approach of KeyInst can achieve better performance compared to pipeline. However, the advantage of pipeline lies in their cost-effectiveness and flexibility. Pipeline can be combined with more powerful LLMs without requiring extensive computational resources and time-consuming training processes.

4.2.2 Comparison of KeyInst-FT and KeyInst-ICL

Table 3 presents the performance of KeyInst-FT and KeyInst-ICL on more powerful LLMs. Using KeyInst-FT to prompt GPT4 achieves the highest EX result at 84.3%. This outcome demonstrates the advantage of the pipeline approach, which can achieve excellent performance with low computational resources by leveraging the powerful natural language processing capabilities of LLMs.

Table 3 also highlights the superior performance of KeyInst-FT over KeyInst-ICL. KeyInst-ICL provides overly detailed keyword suggestions, such as `AVG`, `COUNT`, `JOIN`, and `IN` (see Appendix D for examples). These keywords, defined as the lowest priority in §2, are not expected to appear in KeyInst. Excessive detail can hinder LLMs'

Methods	EX
The single-pass approach	
KeyLla	79.1
w/o question analysis	72.9
w/o keyword suggestion	68.4
The pipeline approach	
KeyInst-FT + GPT4	84.3
w/o question analysis	83.4
w/o keyword suggestion	81.5

Table 4: Ablation study.

Type	DIN-SQL	DAIL-SQL	SC-SQL	KeyInst-FT
GROUP BY	75.3	77.3	76.7	**78.7**
HAVING	85.3	85.3	84.0	**86.0**
ORDER BY	**94.7**	94.0	93.3	94.0
LIMIT	88.7	89.3	88.0	**89.3**
EXCEPT	75.3	82.0	78.7	**83.3**
INTERSECT	75.3	84.0	79.7	**85.3**
UNION	62.0	62.7	65.3	**73.3**
Overall	79.5	82.1	81.0	**84.3**

Table 5: Execution accuracy (EX) of GPT4 using baselines and KeyInst-FT on StrucQL.

Methods	Spider	Bird
Zero-shot	77.9	43.6
DIN-SQL	85.1	50.7
DAIL-SQL	83.1	**54.8**
SC-SQL	**86.2**	53.3
KeyInst-FT	82.8	50.1
+ DIN-SQL	86.8	54.5
+ DAIL-SQL	85.2	**58.0**
+ SC-SQL	**87.6**	56.6

Table 6: Execution accuracy (EX) of GPT4 on the Spider dev and Bird dev.

Text-to-SQL performance (Tai et al., 2023; Tan et al., 2024), which may explain KeyInst-ICL's slightly poorer results. Additionally, KeyInst-FT is generated by a Llama3-8B model fine-tuned on over 6200 KeyInst data points, while KeyInst-ICL is generated by prompting GPT-4 with a 6-shot prompt. Although GPT4 is more powerful, fine-tuning enables the fine-tuned KeyInst generator to better capture the relationship between NLQ and KeyInst, thereby generating KeyInsts more suitable for the Text-to-SQL task.

4.2.3 Ablation Study

Table 4 presents the results of the ablation study. Each KeyInst consists of two parts: question analysis and keyword suggestion (see Figure 2). The question analysis explains the NLQ of the current Text-to-SQL task, while the keyword suggestion provides potential SQL keywords for the current Text-to-SQL task. To assess the contribution of each component, we compared single-pass and pipeline approaches both with and without these parts. For the single-pass approach, we split the training data accordingly, and for the pipeline approach, we separated the KeyInst components when prompting the LLMs. The results in Table 4 indicate that the keyword suggestion plays a more significant role in the effectiveness of KeyInst.

4.2.4 Results of Baselines

Table 5 presents the performance of baseline methods (DIN-SQL (Pourreza and Rafiei, 2024a), DAIL-SQL (Gao et al., 2023), SC-SQL (Tan et al., 2024)) and our KeyInst-FT method on StrucQL. The results indicate that while these baseline methods, as SOTA Text-to-SQL prompting approaches, achieve commendable results on well-known benchmarks (e.g., Spider and Bird), there is still room for improvement in SQL formulation capabilities, particularly in set operations (`EXCEPT`, `INTERSECT`, `UNION`), where our method excels. We believe that explicitly mentioning SQL keywords relevant to the current Text-to-SQL task in the prompt is crucial for enhancing the LLMs' SQL formulation performance. This is supported by DAIL-SQL's strong performance, which is attributed to its consideration of SQL skeleton similarity when constructing few-shot prompts, thus the prompt may contain important SQL keywords that are relevant to the current task.

4.3 Results on General Benchmark

We also evaluated KeyInst on well-known benchmarks, such as Spider and Bird. KeyInst is instruction and can be easily integrated with existing Text-to-SQL prompting methods by appending KeyInst to their prompts (see Appendix E for examples). We used the GPT4 model to assess the performance of baseline methods with KeyInst, with results shown in Table 6.

We conducted experiments using KeyInst-FT (a variant of KeyInst). When used alone, KeyInst serves as a zero-shot prompting method. While it performs well compared to standard zero-shot methods, it does not match the current these SOTA Text-to-SQL prompting methods because it specifically addresses the SQL formulation challenge and does not focus on the schema linking challenge. However, this issue is easily resolved when combined with SOTA methods, which handle schema linking while KeyInst focuses on SQL formulation.

Table 6 demonstrates significant performance improvements in baseline methods after incorporating KeyInst, highlighting KeyInst's effectiveness in SQL formulation. Additionally, KeyInst's ease of integration with prompting methods makes it a valuable tool for advancing prompt-based Text-to-SQL research.

4.4 Discussion

How to choose the application method for KeyInst? We propose two applications for KeyInst: single-pass and pipeline. When computational resources are abundant, the single-pass approach, which involves fine-tuning a model with KeyInst can maximize its SQL formulation capabilities. However, since computational resources are often limited, the pipeline approach becomes more advantageous as it can leverage more powerful models without extensive training. Therefore, we believe that the pipeline approach for KeyInst deserves more attention. Within this approach, fine-tuning a KeyInst generator (if resources allow) can produce more effective KeyInsts than using the in-context learning KeyInst generator.

How to use KeyInst? Due to KeyInst's lightweight design, KeyInst offers strong compatibility, especially evident in the prompting method. We do not recommend relying solely on KeyInst to solve Text-to-SQL tasks, as these tasks often also encounter the challenge of schema linking. We advocate for the integration of KeyInst with other Text-to-SQL prompting methods. This integration is straightforward because KeyInst is presented as an instruction within a prompt. The excellent compatibility of KeyInst holds significant potential for future research.

5 Relate Work

SQL formulation Previous works typically propose well-designed decoders to address SQL formulation challenge. (Wang et al., 2020; Cai et al., 2021; Qi et al., 2022). RESDSQL (Li et al., 2023) introduces a skeleton-aware decoder that first generates an SQL skeleton and then fills the slots, proving to be very effective. A new trend involves prompting LLMs (Chen et al., 2023; Liu et al., 2023), focusing on task decomposition, or selecting demonstrations for few-shot prompts. DIN-SQL (Pourreza and Rafiei, 2024a) uses a pipeline to sequentially address schema linking and SQL formulation, while SC-SQL (Tan et al., 2024) emphasizes result consistency (Wang et al., 2022). DAIL-SQL (Gao et al., 2023) selects demonstrations based on masked question and SQL skeleton similarity. Nan (Nan et al., 2023) and Guo (Guo et al., 2023) propose similar methods. These approaches rely on implicit information in demonstrations, leading to suboptimal SQL formulation. In contrast, KeyInst explicitly guides LLMs to use specific SQL keywords through tailored instructions.

Benchmarks Popular benchmarks like Spider (Yu et al., 2018), and Bird (Li et al., 2024b) evaluate comprehensive Text-to-SQL capabilities. More challenging datasets like Spider-Syn (Gan et al., 2021a), Spider-DK (Gan et al., 2021b), and Ambiguity (Bhaskar et al., 2023) focus on schema linking. However, the field lacks a benchmark for SQL formulation performance. Therefore, we propose StrucQL to help researchers assess the SQL formulation capabilities of Text-to-SQL systems.

6 Conclusion

This paper introduces KeyInst, a dynamic instruction method explicitly highlighting essential SQL keywords likely to be included in the target SQL query. We explore two approaches for integrating KeyInst into Text-to-SQL parsing: the pipeline approach and the single-pass approach. In the pipeline approach, KeyInst is used to prompt LLMs. In contrast, the single-pass approach involves fine-tuning a model with KeyInst. Our results indicate that, for models of the same size, the single-pass approach outperforms the pipeline approach. However, the pipeline approach excels in flexibility, easily integrating with more powerful LLMs to achieve superior performance. Due to KeyInst's lightweight design, KeyInst integrates seamlessly with existing Text-to-SQL prompting methods, enhancing their performance. This compatibility suggests promising potential for future research in Text-to-SQL prompting.

Limitations

In this paper, we made an effort to demonstrate the effectiveness of KeyInst, but there are still some limitations that need to be noted: First and foremost, we acknowledge that KeyInst, designed for the SQL formulation challenge, offers limited assistance for the schema linking challenge. Whether this instruction-based method can be effectively used to address the schema linking challenge requires further exploration in the future. Second, when discussing applications for KeyInst (§4.2.1), due to budget constraints, we conducted experiments only on Llama3-8B. We are uncertain about the performance of the single-pass approach on larger models. Third, prompting with KeyInst has shown excellent compatibility, as it can be combined with other prompting methods and enhance their performance. However, for fine-tuning with KeyInst, it remains unclear whether using KeyInst for fine-tuning existing Text-to-SQL models (Pourreza and Rafiei, 2024b; Li et al., 2024a) will improve their performance. This requires further investigation in future research.

References

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improving text-to-sql generation under ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7053–7074.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems*, 34:7664–7676.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *The 2021 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515.

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

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 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. Prompting gpt-3.5 for text-to-sql with desemanticization and skeleton retrieval. In *Pacific Rim International Conference on Artificial Intelligence*, pages 262–274. Springer.

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

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data.*

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

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*.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14935–14956.

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. *arXiv preprint arXiv:2302.08468*.

Zheng Ning, Yuan Tian, Zheng Zhang, Tianyi Zhang, and Toby Jia-Jun Li. 2024. Insights into natural language database query errors: From attention misalignment to user handling strategies. *ACM Transactions on Interactive Intelligent Systems*.

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

Mohammadreza Pourreza and Davood Rafiei. 2024b. Dts-sql: Decomposed text-to-sql with small large language models. *Preprint*, arXiv:2402.01117.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, 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*.

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

Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain-of-thought style prompting for text-to-sql. *arXiv preprint arXiv:2305.14215*.

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

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Caiming Xiong, et al. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 3911–3921. Association for Computational Linguistics.

A Prompt of Question Analysis

In §2, we constructed a KeyInst set, where the KeyInsts were pre-prepared. For the question analysis part of the KeyInsts, we used the following prompt to guide the GPT4 model. It is a few-shot prompt containing 7 demonstrations. The prompt is:

Please analyse the following natural language query.
Natural language query: Please show the different statuses of cities and the average population of cities with each status.
Analysis: The question is asking for a list of different statuses of cities and the average population for cities within each status. This requires grouping the cities by their status and calculating the average population for each group.

Please analyse the following natural language query.
Natural language query: What is the average longitude of stations that never had bike availability more than 10?
Analysis: The question is looking to calculate the average longitude of bike stations where the number of available bikes never exceeded 10. This requires filtering out stations based on a condition applied to their bike availability data.

Please analyse the following natural language query.
Natural language query: List the writers of the books in ascending alphabetical order.
Analysis: The question is asking to retrieve a list of writers from the book table and sort them in ascending alphabetical order. This requires selecting the Writer column and ordering the results.

Please analyse the following natural language query.
Natural language query: List the publisher of the publication with the highest price.
Analysis: The question is asking to identify the publisher of the publication that has the highest price. This requires sorting the publications by price in descending order and selecting the top result.

Please analyse the following natural language query.
Natural language query: Show ids for all employees who don't have a certificate.
Analysis: The question is asking for the IDs of employees who do not possess any certificates. This requires comparing two sets of data: one from the Employee table and one from the Certificate table, and then finding the difference between these two sets.

Please analyse the following natural language query.
Natural language query: Show names for all employees who have certificates on both Boeing 737-800 and Airbus A340-300.
Analysis: The question is looking for the names of employees who hold certificates for both the Boeing 737-800 and the Airbus A340-300 aircraft. This requires identifying employees who have certificates for both aircraft types and then retrieving their names.

Please analyse the following natural language query.
Natural language query: Find courses that ran in Fall 2009 or in Spring 2010.
Analysis: The question is looking for courses that were offered either in the Fall semester of 2009 or in the Spring semester of 2010. This requires filtering records based on specific conditions for both the semester and the year.

B Comparsion of KeyInst and SQL skeleton.

In §2, we assigned different priorities to SQL keywords and considered which keywords should be included in KeyInst's keyword suggestions based on these priorities. This approach was taken because general keywords (e.g., `JOIN`, `IN`, `COUNT`) do not directly reflect the query intent corresponding to NLQs. Instead, overly detailed information can increase the burden on the KeyInst generator and potentially affect the output of LLMs.

We conducted a comparative experiment where we did not set keyword priorities. In this scenario, KeyInst degraded into an SQL skeleton, as illustrated in Figure 3. The details of the comparative experiment are as follows: we replaced the KeyInst in the KeyInst set with SQL skeletons and fine-tuned the Llama3-8B model to become an SQL skeleton generator. This generator produces an SQL skeleton for each Text-to-SQL task, and this generated SQL skeleton is then used as part of the prompt to guide the LLMs in generating SQL.

The results in Table 7 show that using the skeleton is significantly less effective than using KeyInst-FT(a version of KeyInst). Although the skeleton (Figure 3) appears more specific on the surface compared to KeyInst, directly deriving an SQL skeleton from an NLQ is not easy. This often leads to unexpected errors, which can mislead the LLMs when generating SQL.

Question

What is the name of all the people who are older than at least one engineer? Order them by age.

KeyInst	**SQL Skeleton**
The question is looking for the names of all people who are older than at least one person with the job title 'engineer'. This requires comparing ages and filtering based on a subquery. To translate this question into SQL, you may need to use the following SQL keywords: '**WHERE**', '**ORDER BY**'.	The question is looking for the names of all people who are older than at least one person with the job title 'engineer'. This requires comparing ages and filtering based on a subquery. To translate this question into SQL, you may need to use the following SQL skeleton: **SELECT _ FROM _ WHERE _ > (SELECT min(_) FROM _ WHERE _ = _) OREDER BY _**

Gold SQL

SELECT name FROM Person **WHERE** age >
(SELECT min(age) FROM person **WHERE** job = 'engineer') **ORDER BY** age

Figure 3: Examples of KeyInst-FT and SQL skeleton.

Type	Skeleton	KeyInst-FT
GROUP BY	77.3	**78.7**
HAVING	85.3	**86.0**
ORDER BY	92.7	**94.0**
LIMIT	87.3	**89.3**
EXCEPT	76.0	**83.3**
INTERSECT	78.7	**85.3**
UNION	60.0	**73.3**
Overall	79.6	**84.3**

Table 7: Execution accuracy (EX) of GPT4 using SQL skeleton and KeyInst-FT on StrucQL.

C An Example of Schema Simplification.

We constructed the StrucQL benchmark to intuitively evaluate a Text-to-SQL system's SQL formulation performance by minimizing schema linking's impact. This is achieved through schema simplification, which aims to reduce the complexity of schema linking in Text-to-SQL tasks, thereby decreasing the errors caused by incorrect schema links. Figure 4 provides an example of the schema-simplified question and schema. Specifically, we marked and modified schema-related words in the question. For instance, in Figure 4, 'name of the shop' was changed to 'shop.name'. Additionally, we filtered out tables and columns from the database schema that are irrelevant to the current question.

	Original Instacne	Schema-simplified Instance
Question	For each shop, return the number of employees working there and the name of the shop.	For each \`shop\`, return the number of employees working there and the \`shop.name\`.
Schema	A SQLite database 'employee_hire_evaluation' has 4 tables named 'employee, shop, hiring, evaluation'. employee table has columns: 'Employee_ID, Name, Age, City'. shop table has columns: 'Shop_ID, Name, Location, District, Number_products, Manager_name'. hiring table has columns: 'Shop_ID, Employee_ID, Start_from, Is_full_time'. evaluation table has columns: 'Employee_ID, Year_awarded, Bonus'.	A SQLite database 'employee_hire_evaluation' has 2 tables named 'hiring, shop'. hiring table has columns: 'shop_id'. shop table has columns: 'shop_id, name'."

Figure 4: An example of Schema-Simplifed question and schema.

D Examples of KeyInst-FT and KeyInst-ICL

We propose two variants of KeyInst: KeyInst-FT and KeyInst-ICL KeyInst-FT is generated by a fine-tuned Llama3-8B model, while KeyInst-ICL is generated by guiding LLMs using In-Context learning. Our experiments demonstrate that KeyInst-FT performs better. Examples of KeyInst-FT and KeyInst-ICL are provided in Figure 5. These examples show that KeyInst-FT aligns more closely with the requirements of gold SQL. Specifically, KeyInst-FT consistently produces more accurate and contextually appropriate keyword suggestions. This comparison highlights the advantage of fine-tuning models for specific tasks.

Question

How many total credits are offered by each department?

KeyInst-FT	KeyInst-ICL
The question is asking for the total number of credits offered by each department. This requires aggregating the \`course.credits\` for each \`course.dept_name\`. To translate this question into SQL, you may need to use the following SQL keywords: '**GROUP BY**'.	The question is asking for the total number of course credits offered by each department. This requires summing up the credits for courses grouped by the department name. Suggested SQL keyword To translate this question into SQL, you may need to use the following SQL keywords: '**GROUP BY**', '**SUM**'.

Gold SQL

SELECT sum(credits) , dept_name FROM course **GROUP BY** dept_name

Question

What are the names of customers who never made an order.

KeyInst-FT	KeyInst-ICL
The question is looking for the names of customers who have never placed an order. This requires identifying customers who do not appear in the list of customers who have made an order. The SQL keywords needed will involve set operations to exclude customers who have placed orders from the list of all customers. To translate this question into SQL, you may need to use the following SQL keywords: '**EXCEPT**'.	The question is asking for the names of customers who have never placed an order. This requires comparing the data from the "customers" table with the data from the "orders" table and identifying customers who do not appear in the "orders" table. Suggested SQL keyword To translate this question into SQL, you may need to use the following SQL keywords: '**LEFT JOIN**', '**WHERE**', '**IS NULL**'.

Gold SQL

SELECT customer_name FROM customers
EXCEPT
SELECT t1.customer_name FROM customers AS t1 JOIN customer_orders AS t2
ON t1.customer_id = t2.customer_id

Figure 5: Examples of KeyInst-FT and KeyInst-ICL.

E The usage of KeyInst

KeyInst is represented as a single instruction, which gives it excellent compatibility and allows it to integrate with existing Text-to-SQL prompting methods seamlessly. Specifically, each KeyInst is tailored to the current Text-to-SQL task. To use it, we place it with the current Text-to-SQL task, typically at the end of the prompt. Figure 6 shows examples of using KeyInst. Note that in the few-shot prompt, we did not add KeyInst to each demonstrations, it is solely intended for the current Text-to-SQL task.

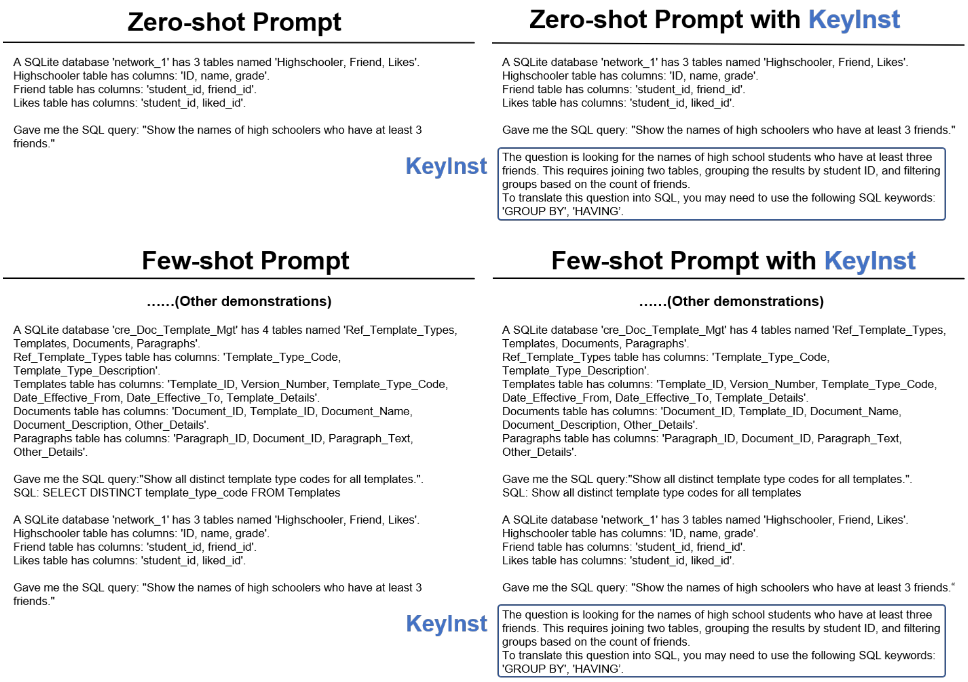

Figure 6: Examples of using KeyInst in zero-shot prompt and few-shot prompt.

F Performance of LLMs with KeyInst on StrucQL.

We evaluated the performance of various LLMs on StrucQL after using KeyInst (KeyInst-FT). The results in Table 8 show that KeyInst is a simple and effective method that significantly enhances the SQL formulation performance of LLMs. Notably, Llama3-70B with KeyInst is only 1.1% behind GPT-4 with KeyInst.

Type	Gemma-7B	Llama3-8B	Llama3-70B	Claude3	GPT4
		Zero-shot			
GROUP BY	58.7	69.3	**78. 0**	73.3	76.7
HAVING	52.7	74.0	82.0	85.3	**86.0**
ORDER BY	66.7	74.7	90.7	92.7	**94.0**
LIMIT	51.3	76.0	80.7	77.3	**88.7**
EXCEPT	41.3	63.3	64.7	70.7	**72.7**
INTERSECT	42.7	57.3	61.3	**73.3**	72.7
UNION	30.7	38.7	47.3	55.3	**57.3**
Overall	49.1	63.9	72.0	75.4	**78.3**
		Zero-shot with KeyInst-FT			
GROUP BY	61.3	74.5	**82.0**	78.0	78.7
HAVING	58.0	80.0	82.0	**88.0**	86.0
ORDER BY	71.3	88.0	92.7	93.3	**94.0**
LIMIT	60.7	82.7	86.0	80.7	**89.3**
EXCEPT	45.3	69.3	86.7	**86.7**	83.3
INTERSECT	56.7	81.3	84.7	85.3	**85.3**
UNION	44.7	62.0	68.7	**76.7**	73.3
Overall	56.9	76.9	83.2	84.1	**84.3**

Table 8: Execution accuracy results for all compared LLMs on StrucQL after using KeyInst-FT.