2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00009

PURPLE: Making a Large Language Model a Better SQL Writer

Tonghui Ren[†], Yuankai Fan[†], Zhenying He[†], Ren Huang[†], Jiaqi Dai[†], Can Huang[†],
Yinan Jing[†], Kai Zhang[†], Yifan Yang[‡], X.Sean Wang[†]
[†]*School of Computer Science, Fudan University*
[‡]*Transwarp Technology (Shanghai) Co., Ltd*
thren22@m.fudan.edu.cn, {fanyuankai, zhenying}@fudan.edu.cn, {renhuang21, daijq22, huangcan22}@m.fudan.edu.cn,
{jingyn, zhangk}@fudan.edu.cn, yifan.yang@transwarp.io, xywangCS@fudan.edu.cn

***Abstract*—Large Language Model (LLM) techniques play an increasingly important role in Natural Language to SQL (NL2SQL) translation. LLMs trained by extensive corpora have strong natural language understanding and basic SQL generation abilities without additional tuning specific to NL2SQL tasks. Existing LLMs-based NL2SQL approaches try to improve the translation by enhancing the LLMs with an emphasis on user intention understanding. However, LLMs sometimes fail to generate appropriate SQL due to their lack of knowledge in organizing complex logical operator composition. A promising method is to input the LLMs with demonstrations, which include known NL2SQL translations from various databases. LLMs can learn to organize operator compositions from the input demonstrations for the given task. In this paper, we propose PURPLE (Pre-trained models Utilized to Retrieve Prompts for Logical Enhancement), which improves accuracy by retrieving demonstrations containing the requisite logical operator composition for the NL2SQL task on hand, thereby guiding LLMs to produce better SQL translation. PURPLE achieves a new state-of-the-art performance of 80.5% exact-set match accuracy and 87.8% execution match accuracy on the validation set of the popular NL2SQL benchmark Spider. PURPLE maintains high accuracy across diverse benchmarks, budgetary constraints, and various LLMs, showing robustness and cost-effectiveness.**

***Index Terms*—NLIDB, NL2SQL, SQL, LLMs**

I. Introduction

The task of Natural Language to SQL (NL2SQL) translation helps the Database Management Systems (DBMS) be more user-friendly. The NL2SQL approach translates Natural Language (NL) query into SQL based on the database, enabling users to easily access data in a DBMS without needing knowledge of the database schema or SQL syntax.

Recently, general-purpose Large Language Models (LLMs) have exhibited profound capabilities in various downstream tasks without the need for a costly LLM fine-tuning process, including NL2SQL [1]–[8]. Thanks to the strong NL understanding ability, existing approaches can achieve high Execution Match[1] accuracy. For example, DIN-SQL [2] is one of the state-of-the-art (SOTA) approaches based on a few-shot Chain-of-Thought (CoT) strategy [9], which can achieve 82.8% execution match accuracy on the validation set of the NL2SQL benchmark, Spider [10].

TABLE 1: LLMs-based approaches accuracy on Spider.

Strategy	Exact-Set Match%	Execution Match%
ChatGPT-SQL	37.9	70.1
C3	43.1	81.8
DIN-SQL(GPT4)	60.1	82.8
DAIL-SQL(GPT4)	**68.7**	83.6

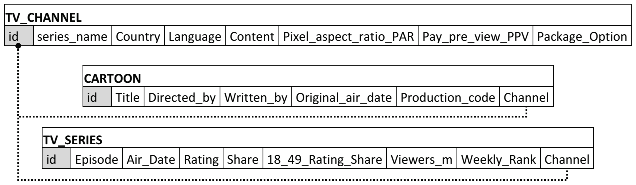
Upon analyzing the translations of existing LLMs-based approaches, we observe that they achieve high execution accuracy thanks to the strong NL understanding ability of the LLMs, while the LLMs only have basic SQL knowledge for SQL writing. We notice that all of the existing LLMs-based NL2SQL approaches fail to achieve high Exact-Set Match[2] accuracy as shown in Table 1, which is more rigorous compared to execution match accuracy. The SQL queries with the same execution result may have different semantics, which means execution match accuracy will overestimate the performance of approaches, leading to the false positive [10]. The complexity of SQL is mainly from the **logical operator composition**, which is not what general LLMs are good at.

Existing works apply zero-shot or few-shot strategies to enhance LLMs with task-specific knowledge. Zero-shot approaches, such as C3 [11], employ instructional prompts to guide the utilization of SQL keywords. On the other hand, DAIL-SQL [12] and DIN-SQL [2] are two few-shot strategies to improve the capabilities of LLMs through few-shot learning [13]. Both DAIL-SQL and DIN-SQL emphasize the importance of NL understanding. DAIL-SQL integrates SQL keyword similarity for demonstration selection. However, they fail to provide the knowledge of operator composition in SQL formulation. The LLMs understand the user intention but lack related knowledge in organizing logical operator composition for SQL generation, resulting in a semantic similar but incorrect SQL. Figure 1 illustrates a case of NL2SQL task. The abovementioned approaches implement the "*NOT IN*" operator, corresponding to the "*not playing*" in the NL query. Both C3 and DAIL-SQL do capture the meaning of excluding "*countries*" but failed to implement it because such a semantic needs a "*JOIN*" operator in the SQL. DIN-SQL generates a likely correct prediction but fails to recognize that the "*EXCEPT*" keyword involves a de-duplication operation,

[1] Execution Match: SQL equivalence based on the execution result.

[2] Exact-Set Match: SQL equivalence at the SQL component level.

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00009



(a) Database schema from Spider for the example.

NL Query:	*What are the **countries** that are not playing **cartoons** written by **Todd Casey**?*
	C3 (ChatGPT)

Generated SQL:
(Incorrect)
	`SELECT TV_CHANNEL.Country FROM TV_CHANNEL WHERE`
	`TV_CHANNEL.id NOT IN (SELECT CARTOON.channel FROM CARTOON`
	`WHERE CARTOON.Written_by='Todd Casey'`**`)`**`;`

	DAIL-SQL (GPT4)

Generated SQL:
(Incorrect)
	`SELECT Country FROM TV_CHANNEL WHERE` **`id NOT IN (`**
	`SELECT Channel FROM CARTOON` `WHERE Written_by='Todd Casey'`**`)`**`;`

	DIN-SQL (GPT4)

Generated SQL:
(Incorrect)
	`SELECT Country FROM TV_CHANNEL WHERE Country` **`NOT IN (`**
	`SELECT T1.Country FROM TV_CHANNEL AS T1 JOIN CARTOON AS T2`
	`ON T1.id = T2.Channel WHERE T2.Written_by='Todd Casey'`**`)`**`;`

Gold SQL:	`SELECT Country FROM TV_CHANNEL`
	`EXCEPT`
	`SELECT T1.Country FROM TV_CHANNEL AS T1 JOIN CARTOON AS T2`
	`ON T1.id = T2.Channel` `WHERE T2.Written_by='Todd Casey';`

(b) NL query from Spider and the corresponding translation result from different approaches.

Fig. 1: An example of NL2SQL translation task from Spider.

resulting in redundant outcomes. **Despite the three SOTA LLM-based approaches capturing user intentions, they failed in managing complex logical operator compositions.** Such as the necessity for a "*JOIN*" operator or in distinguishing the difference between "*NOT IN*" and "*EXCEPT*" in SQL.

In this study, we aim to enhance the SQL generation capabilities of general LLMs on NL2SQL tasks, making an LLM a better SQL writer. We hope that such an approach can achieve high execution accuracy by leveraging the robust NL comprehension inherent to LLMs, as well as high exact-set match accuracy to maintain logical semantic integrity. The **main challenge** is to provide requisite logical composition knowledge without exceeding the input length budget. Given the limited input length and the infinite potential logical compositions, it is impractical to contain all composition knowledge within the prompt.

To enhance the LLMs with corresponding SQL logical operator composition knowledge within the limited input length, we introduce PURPLE, Pre-trained models Utilized to Retrieve Prompts for Logical Enhancement, a novel few-shot prompting strategy tailored for LLMs-based NL2SQL translation. The key point of PURPLE is the **demonstration selection**, which needs to select the demonstrations containing the requisite logical operator composition. The demonstrations[3] are NL2SQL tasks derived from various databases, each containing an NL query, database information, and SQL translation. LLMs can learn from the demonstrations in the prompt, which involves the selected demonstrations and the description of the current NL2SQL task, about how to handle the NL2SQL task, especially managing the operator composition, which is challenging for LLMs. We employ a fine-tuned model to identify the logical operator compositions knowledge relevant to the current task. Moreover, we introduce a demonstration selection strategy based on the inferred knowledge. This approach is for both generalization and fuzzification, considering the limited size of all demonstrations and the capabilities of the fine-tuned prediction model.

PURPLE consists of four main modules: **Schema Pruning**, **Skeleton Prediction**, **Demonstration Selection**, and **Database Adaption**. Initially, PURPLE employs a classifier and a probability-based algorithm to prune irrelevant schema items for a given NL query. Subsequently, the pruned schema is used to infer a SQL skeleton, which masks all database-specific values compared with SQL, that contains the requisite operator composition knowledge. PURPLE retrieves relevant demonstrations based on the inferred skeleton. Following the LLM calling, PURPLE adjusts the output to adapt to the specific database schema and SQL dialect, thereby mitigating the LLM-induced hallucination problems.

To show the performance of PURPLE, we conduct a comprehensive evaluation of our strategy from multiple perspectives on four mainstream benchmarks. Moreover, we explore the trade-off between cost and performance. Notably, PURPLE is flexible because it can be configured for higher performance at a higher cost or optimized for reducing the expense of some performance drop. We further compare various approaches across different LLMs to evaluate the performance fluctuation. An ablation study is also conducted to show the effectiveness of each module in PURPLE.

The contributions of this paper are summarized as follows:

- We propose PURPLE, a novel approach leveraging pretrained models to generate optimized prompts for LLMs and augment the performance of NL2SQL translation.
- We enhance the SQL writing ability of LLMs by selecting demonstrations containing the requisite operator composition knowledge, helping the LLMs perform better.
- We conceptualize the SQL logical composition knowledge through an automaton framework, defining four levels of automaton state abstraction. This modeling helps select the valuable demonstration for PURPLE.
- We test PURPLE through comprehensive experiments. The outcomes show superior performance, especially an 11.8% improvement in exact-set match accuracy compared to the existing LLMs-based NL2SQL approaches. The experiment also shows the robustness and cost-effectiveness of PURPLE.

The paper is organized as follows: Section II introduces essential preliminaries. Section III gives an overview of PURPLE. The core modules are explained in Section IV. Experimental results are discussed in Section V. Related works and conclusions are shown in Section VI and Section VII.

II. Preliminaries

NL2SQL translation has benefited from advancements in Natural Language Processing (NLP). This section provides the foundational concepts and definitions relevant to this study.

[3] A detailed description of demonstrations is shown in Section III-A.

A. Language Models

A language model (LM) is a statistical model fundamental to many NLP tasks, typically trained on extensive text corpora. LMs have been applied to various tasks, including NL2SQL. We categorize LMs as PLMs and LLMs in this paper.

PLMs: PLMs refer to LMs with a relatively smaller parameter size in this paper, which can not applied to downstream tasks without fine-tuning. Limited by their model size and pre-training corpus, these models do not exhibit capabilities that can be directly applied to downstream tasks. Notable examples of PLMs include BERT [14], BART [15], and T5 [16].

LLMs: This category refers to LMs with a huge parameter size demonstrating ability across many downstream tasks without tuning. Instruction design or in-context learning can be employed to adapt LLMs to different tasks. Such models include GPT3 [17], PaLM [18], ChatGPT, and GPT4.

B. LLMs-based NL2SQL

The abilities of LLMs for NL understanding and generation have drawn attention from researchers. We categorize LLMs-based NL2SQL approaches into zero-shot and few-shot. Both approaches enhance the performance of LLMs on downstream tasks through prompts, which are sequences of textual instructions that elicit outputs from LLMs.

For a typical NL2SQL translation task, the input consists of an NL query $\mathcal{X}$ and database information $\mathcal{D}$. The goal is to obtain the target SQL $\mathcal{Y}$. The task can be formulated as:

$$\hat{\mathcal{Y}} = LLM(P(\mathcal{X}, \mathcal{D}, \mathcal{E}))$$

In this function, LLM denotes the LLM call, P denotes the prompt generation, and $\mathcal{E}$ denotes known NL2SQL translation that can be used as auxiliary information for the LLMs.

Zero-shot: Zero-shot NL2SQL translation does not include annotated examples. In this context, P can be represented as:

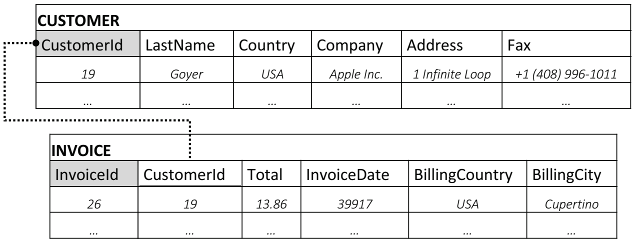
$$P_0(\mathcal{X}, \mathcal{D}, \varnothing)$$

The prompt generation relies solely on the information of the current translation task. C3 [11] and ChatGPT-SQL [5] are two typical zero-shot NL2SQL approaches.

Few-shot: With a few demonstrations from the annotated datasets, the LLMs can learn how to generate the correct SQL. We consider the training set of the benchmark as the source of demonstrations, maintaining the cross-domain setting. The prompt generation process can be represented as:

$$P_f(\mathcal{X}, \mathcal{D}, \mathcal{E})$$

The $\mathcal{E}$ is the demonstrations from annotated datasets, detailed descriptions will be formally outlined in section III-A.



CUSTOMER

CustomerId	LastName	Country	Company	Address	Fax
19	*Goyer*	*USA*	*Apple Inc.*	*1 Infinite Loop*	*+1 (408) 996-1011*
...	...	...	...	...	...

INVOICE

InvoiceId	CustomerId	Total	InvoiceDate	BillingCountry	BillingCity
26	*19*	*13.86*	*39917*	*USA*	*Cupertino*
...	...	...	...	...	...

(a) Database information for a demonstration

NL:	*What are the last names of customers without invoice totals exceeding 20?*

SQL:
(Skeleton)
	`SELECT LastName FROM CUSTOMER`
	`EXCEPT`
	`SELECT T1.LastName FROM CUSTOMER AS T1 JOIN Invoice AS`
	`T2 ON T1.CustomerId = T2.CustomerId WHERE T2.total > 20`

(b) NL query and SQL for a demonstration

Fig. 2: An example for demonstrations

C. SQL skeleton

In this study, we introduce the concept of a SQL skeleton, denoted as $\mathcal{S}$. The skeleton serves as a structural template abstracting from database-specific details, thereby focusing on the logical operator composition inherent within SQL queries. The skeleton preserves all operational keywords while substituting placeholders for specific database elements like tables, columns, and constant values. For instance, the SQL skeleton of the gold SQL in Figure 1b is:

```
SELECT _ FROM _
EXCEPT
SELECT _ FROM _ JOIN _ ON _ = _ WHERE _ = _
```

This abstraction focuses on the operational logic of the SQL, providing a generalized yet structurally representative form.

III. Method Overview

In this section, we explore the demonstrations as an input source for PURPLE and present an overview of the pipeline.

A. Demonstration

In the context of few-shot LLMs-based NL2SQL translation, demonstrations serve as examples that LLMs can learn to handle the current task. Each demonstration consists of task inputs and corresponding outputs. Based on cross-database settings in this paper, we employ the original training data from the NL2SQL benchmarks as demonstrations.

Specifically, a demonstration $e_i \in \mathcal{E}$ contains three components: the NL $\mathcal{X}^{e_i}$, the database $\mathcal{D}^{e_i}$, and the target SQL $\mathcal{Y}^{e_i}$. Figure 2 provides an illustrative example of a demonstration. The $\mathcal{D}^{e_i}$ includes the database schema and the data. We select a subset of representative values for each column like [19] to optimize the length of database information. Formally, a demonstration can be represented as:

$$e_i = \mathrm{CAT}(\mathcal{D}^{e_i}, \mathcal{X}^{e_i}, \mathcal{Y}^{e_i})$$

Here, CAT represents the string concatenation. The prompt structure within PURPLE is formulated as:

$$P_f = \mathrm{CAT}(\mathcal{E}', \mathcal{D}, \mathcal{X})$$

In this expression, $\mathcal{E}'$ represents the subset of demonstrations selected from the entire set $\mathcal{E}$ for constructing the prompt.

Moreover, we incorporate a schema pruning strategy in PURPLE. Accordingly, the schema of each demonstration undergoes a pruning process to reduce its length. Section IV-A provides details of such a module. Thus, the database information for a demonstration is a subset of the whole database.

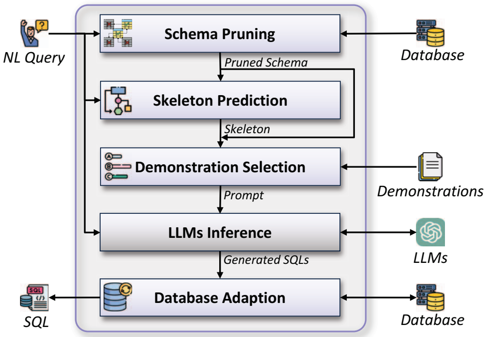


Fig. 3: Overview of PURPLE.

B. *Overview of PURPLE*

The architecture of PURPLE is shown in Figure 3. Firstly, the schema pruning module of PURPLE excludes tables and columns that are not requisite for constructing the target SQL for the current NL query.

The pruned schema and the NL query are used for SQL skeleton prediction. Such a SQL skeleton represents the needed logical composition knowledge required by LLMs. PURPLE selects relevant examples based on the skeleton to form a prompt, which also includes the NL query and the pruned schema for the current NL2SQL task.

PURPLE submits the prompt to LLMs for NL2SQL translation. A database adaption module follows, detecting and fixing hallucination errors induced by the LLMs. PURPLE integrates an execution-consistency [20] strategy into the database adaption module to stabilize the output further. The resulting processed SQL becomes the final output of PURPLE.

1) Schema Pruning: As Step 1 of Figure 3 illustrates, schema pruning narrows down the database information. This module decides which tables or columns are needed for the target SQL based on the NL query and schema. This step prunes the schema to reduce inference complexity for higher translation accuracy, as subsequent modules only process the pruned schema. It is important to keep high recall to reduce the risk of error propagation. We design a pruning strategy based on a trained classifier, trying to keep essential tables or columns while keeping the database information short.

2) Skeleton Prediction: Step 2 of Figure 3 is the skeleton prediction module, which detects the requisite logical composition knowledge for the NL2SQL task. Accurate predictions allow us to extract demonstrations containing essential knowledge. We employ a specialized fine-tuned PLM on the skeleton generation task. The fine-tuning phase equips the PLM with the capability to discern operator compositions. We generate the top-k skeletons by the beam search for high recall.

3) Demonstration Selection: Highlighted in Step 3 of Figure 3, PURPLE selects demonstrations following the predicted SQL skeletons. While it is non-trivial to model the composition knowledge and extract the demonstrations based on the predicted requisite. The selection strategy must have the capacity for generalization to address unseen tasks and incorporate fuzzification to compensate for the limitations of the skeleton prediction model. The complexity of composition knowledge cannot be captured by simplistic similarity functions. We design four levels of SQL skeleton abstraction to facilitate the selection of demonstrations that include composition knowledge for the LLMs. Each higher abstraction level masks more details, focusing on more coarse-grained composition. Such an approach significantly enhances the generalization capabilities of PURPLE for unseen logical composition.

4) Database Adaption: Step 5 in Figure 3 presents the database adaption module. Hallucination issues in LLMs are a common occurrence, often resulting in the generation of buggy SQL queries that are incompatible with specific databases. Unlike methods such as PICARD [21] that employ specialized decoding strategies, we face challenges since we use LLMs as a service. To reduce the buggy SQL generation, we systematically catalog these errors and develop heuristic-based correction algorithms to address them, a low-cost strategy helping LLMs correct the buggy SQL. Such a process can make the output of LLMs fit specific databases, including the specific database schema and specific DBMS SQL dialect. We also include an execution-consistency strategy into PURPLE to stabilize the LLMs generation.

IV. Methodologies

A. *Schema Pruning*

PURPLE begins with a Schema Pruning module, which can be used to eliminate the schema items that will not be used in the target SQL. This module introduces two benefits: Firstly, it shortens the input length for each demonstration, enabling more demonstrations within the token input constraint. Secondly, it simplifies the inference task for LLMs by limiting the problem to a subset of the database schema.

1) Table-Column Classifier: The module takes as input the schema denoted by $\mathcal{D} =< \mathcal{T}, \mathcal{C}, \mathcal{P}, \mathcal{F} >$ and NL query denoted by $\mathcal{X}$. More specifically, $\mathcal{T} = \{t_1, t_2, ..., t_{|\mathcal{T}|}\}$ denote the tables within the database schema. For each table t_i, the columns are denoted by $\mathcal{C}_i = \{c_{i,1}, c_{i,2}, ..., c_{i,|\mathcal{C}_i|}\}$. $\mathcal{P} = \{c_{p_1}, c_{p_2}, ..., c_{p_{|\mathcal{P}|}}\}$ represents the primary keys, and $\mathcal{F} = \{(c_{f_1}, c_{p_1}), (c_{f_2}, c_{p_2}), ..., (c_{f_{|\mathcal{F}|}}, c_{p_{|\mathcal{F}|}})\}$ represents the foreign-primary key pairs.

We implement such a classifier based on the schema ranking module of RESDSQL [22]. The input can be structured as:

$$\mathcal{X}, t_1, c_{1,1}, ..., c_{1,|\mathcal{C}_1|}, ..., t_{|\mathcal{T}|}, c_{|\mathcal{T}|,1}, ..., c_{|\mathcal{T}|,|\mathcal{C}_{|\mathcal{T}|}|}$$

For each t_i and $c_{i,j}$, the classifier will predict whether such table or column is related to the question.

The classifier is trained by the NL2SQL training data. For each input pair of $(\mathcal{X}, \mathcal{D})$, the labels are extracted from the SQL $\mathcal{Y}$ to identify the presence (absence) of each table or column. Training adopts focal loss [23] in line with RESDSQL.

In the inference stage, the classifier yields the probability of relevance for each schema item to the NL query. Tables with a probability exceeding the threshold τ_p are denoted as $\mathcal{T}'$. Similarly, for each table t_i, columns with a probability exceeding τ_p are denoted as $\mathcal{C}'_i$.

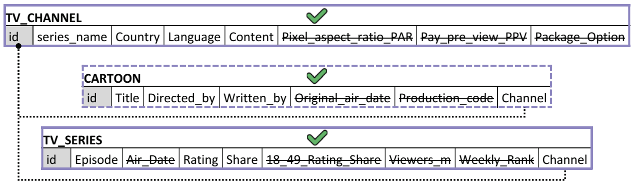


Fig. 4: Schema pruning.

PURPLE adopts a novel method of schema pruning, distinct from the existing methods. The conventional strategy retains the top-k_1 tables and top-k_2 columns, leading to two disadvantages. Firstly, it tends to increase the complexity of schema by including superfluous schema items. Secondly, the selected tables may lack connectivity due to the limited precision of the classifier. These factors necessitate additional processing by LLMs to differentiate among an expanded set of schema items. In contrast, we aim to identify a schema subset that is both closely related and interconnected. PURPLE models the schema pruning task within the framework of a *Steiner Tree Problem* [24], similar to the keywords search studies [25]. We include a redundant boundary to optimize recall.

We represent the schema as a graph $G = (V, E)$, with V representing tables $\mathcal{T}$, and E representing the relationships between them (foreign-primary key connections). Each edge in E is assigned a weight of 1. The tables in $\mathcal{T}'$ as shown in Section IV-A1 can be reduced to the Steiner point set S. So the pruning strategy can be reduced to the *Steiner Tree Problem*, the objective is to extract the smallest connected sub-graph G' containing all tables in $\mathcal{T}'$ from graph G. *Steiner Tree Problem* is an NP-Hard problem. We employ a burst-search algorithm to get the solution thanks to the limited size of the schema currently. Incorporating new algorithms [26] for the larger database is left as future work.

For a high recall to avoid the error propagation problem, the table with the highest probability under τ_p will be included in graph G' if the table has an edge with a node in G'. All nodes in G' are denoted as $\mathcal{T}'$ for the kept tables. For each table t_i in $\mathcal{T}'$, columns with a probability exceeding τ_p and the *primary keys* are kept, denoted as $\mathcal{C}'_i$. We define τ_n as the minimum column number to keep the table semantics.

Following the pruning module, only the target SQL-relevant schema information remains. Any primary keys in $\mathcal{P}$ and foreign keys in $\mathcal{F}$ that are unrelated to the tables $\mathcal{T}'$ and columns $\mathcal{C}'$ will be discarded. For consistency and ease of notation, we continue to denote the pruned schema as $\mathcal{D} =< \mathcal{T}, \mathcal{C}, \mathcal{P}, \mathcal{F} >$.

For the example illustrated in Figure 1, a trained classifier calculates the relevance of each table and column to the NL query. As shown in Figure 4, tables with a probability exceeding τ_p are outlined with a solid purple line. Table *CARTOON* has the highest probability among tables with probabilities below τ_p, marked by a dashed purple line. We include table *CARTOON* for high recall. The columns with a strikethrough will be removed when we set $\tau_n = 5$.

PURPLE focuses on keeping tables that are connected, improving efficiency. It also includes tables likely to be misclassified to boost recall without much extra cost.

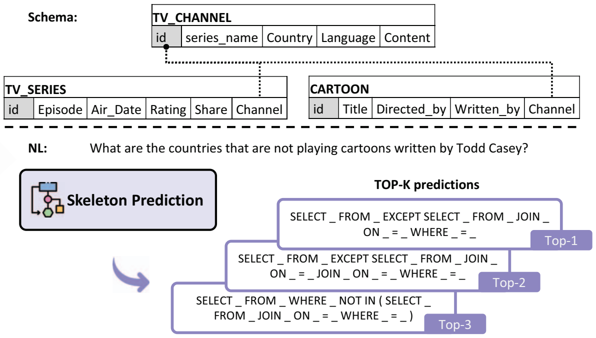

Schema:
TV_CHANNEL
id
series_name
Country
Language
Content
TV_SERIES
id
Episode
Air_Date
Rating
Share
Channel
CARTOON
id
Title
Directed_by
Written_by
Channel
NL:
What are the countries that are not playing cartoons written by Todd Casey?
Skeleton Prediction
TOP-K predictions
SELECT _ FROM _ EXCEPT SELECT _ FROM _ JOIN _ ON _ = _ WHERE _ = _
Top-1
SELECT _ FROM _ EXCEPT SELECT _ FROM _ JOIN _ ON _ = _ JOIN _ ON _ = _ WHERE _ = _
Top-2
SELECT _ FROM _ WHERE _ NOT IN (SELECT _ FROM _ JOIN _ ON _ = _ WHERE _ = _)
Top-3

Fig. 5: Skeleton prediction.

B. Skeleton Prediction

Detecting the requisite operator composition is crucial for acquiring the necessary knowledge for LLMs. We notice that existing PLMs-based approaches achieve high Exact Match accuracy, suggesting that fine-tuning enables PLMs to identify operator compositions. Moreover, we propose a PLMs-based skeleton prediction module. The module uses the top-k predicted skeletons, which have more operator composition diversity than predicted SQL queries. This strategy ensures a high recall of the requisite operator compositions, recognizing that the predicted skeleton is an intermediary rather than the terminal output compared with the PLMs-based approaches.

Our skeleton generator is built on sequence-to-sequence PLMs. The training loss function can be formulated as:

$$\mathcal{L}_{gen} = - \sum_{(\mathcal{X},\mathcal{D},\mathcal{S}) \in Train} \sum_{i=1}^{|\mathcal{S}|} \log P(\mathcal{S}_i | \mathcal{S}_{<i}, \mathcal{X}, \mathcal{D})$$

We process the gold SQL to obtain the target skeleton $\mathcal{S}$ for each training data. Every database-specific entity, including tables, columns, values, and aliases, is replaced by underscores.

We obtain the top-k outputs using beam search [27]. At step i, the skeleton token $\mathcal{S}_i$ is determined by:

$$\mathcal{S}_i = \arg\max_{v \in V} P(v | \mathcal{S}_{<i}, \mathcal{X}, \mathcal{D})$$

V represents the vocabulary of the PLM. The beam search halts upon encountering the stop token. For each $\mathcal{S}$ output, its sequence probability is computed as:

$$P(\mathcal{S}) = \prod_{i=1}^{|\mathcal{S}|} P(\mathcal{S}_i | \mathcal{S}_{<i}, \mathcal{X}, \mathcal{D})$$

We choose T5 [16] as the PLM for skeleton prediction. As illustrated in Figure 5, the top-3 predicted skeletons for the task in Figure 1 are presented. The target skeleton is predicted by the skeleton model as the top-1 output.

The specialized skeleton prediction model has the ability to distinguish requisite composition knowledge due to fine-tuning. This model offers two primary advantages over the PLMs-based NL2SQL models. Firstly, skeleton generation

4. <START> → SELECT → FROM → <IUE> → SELECT → FROM → WHERE → <END>

3. <START> → SELECT → FROM → <IUE> → SELECT → FROM → JOIN → ON → <CMP> → WHERE → <CMP> → <END>

2. <START> → SELECT → FROM → EXCEPT → SELECT → FROM → JOIN → ON → = → WHERE → = → <END>

1. <START> → SELECT → _ → FROM → _ → EXCEPT → SELECT → _ → FROM → _ → JOIN → _ → ON → _ → = → _ → WHERE → _ → = → _ → <END>

Fig. 6: Automaton abstraction example.

abstracts away from SQL details, simplifying the complexity of the task. Secondly, the top-k predictions generated by this model exhibit a higher degree of diversity because the same skeleton with different database-specific tokens are ignored.

C. Demonstration Selection

The main idea of PURPLE is to select a set of demonstrations that contains the necessary logical operator composition, thereby instructing LLMs on generating accurate SQL queries. However, this demonstration-based approach has several challenges when relying on predicted SQL skeletons:

- A selection of demonstrations that precisely match the predicted skeletons will introduce the generalization problem. Given the infinite logical compositions, a finite set of demonstrations will not be enough for all tasks.
- Skeleton prediction accuracy depends on the PLMs used. Even with a top-k strategy for skeleton prediction, achieving complete recall of the target skeleton remains hard. Therefore, enhancing the selection process for potential inaccuracies in the predicted skeleton is important.
- SQL is a complex declarative language that presents difficulties in capturing logical operator composition similarity. Ineffective similarity measures can introduce noise, failing to teach the LLMs to handle the NL2SQL task.

To capture the logical composition knowledge inherent in the demonstrations and to overcome the challenges mentioned above, we propose an automaton-based modeling of SQL composition knowledge with a four-level abstraction hierarchy. An automaton, characterized as a sequence of states, represents a strict operator composition structure. We introduce four-level abstractions to enhance this automaton with generalization and fuzzification capacity. This hierarchical automaton modeling design enables PURPLE to discern the logical operator composition and extract pertinent demonstrations.

1) Automaton Modeling: The sequence of SQL operators, comprising various keywords and their order, conveys distinct semantic compositions. We conceptualize this logical operator composition through a hierarchical abstraction within an automaton framework, thereby encapsulating compositional knowledge across varying granularity. The four discrete abstraction levels of this automaton are named **Detail-Level**, **Keywords-Level**, **Structure-Level**, and **Clause-Level**, each representing a more coarse-grained composition of the SQL query. Figure 6 illustrates an automation abstraction example of the skeleton shown in Section II-C. A detailed description of these levels is as follows:

<AGG> ::= COUNT | MAX | MIN | SUM | AVG
<CMP> ::= < | <= | > | >= | = | != | BETWEEN | NOT LIKE | LIKE | NOT IN | IN
<IUE> ::= INTERSECT | UNION | EXCEPT
<OP> ::= + | - | * | /

Fig. 7: Structure-Level abstraction mapping rules.

1. **Detail-Level**: This level captures each component based on the predicted skeleton. It preserves the placeholders for columns and tables, reflecting the quantity and position of database-related elements.
2. **Keywords-Level**: This level abstracts the placeholders to concentrate on SQL keywords. It contains all SQL keywords to reflect the logical operator composition, such as the comparison operator "**=**". This abstraction level shifts the focus solely to the logical operators within SQL.
3. **Structure-Level**: Specific logical operators are classified under broader categories. For instance, "**=**" is generalized to "**<CMP>**", and "**EXCEPT**" to "**<IUE>**". This abstraction masks the detailed semantics, enabling the automaton to capture the structural semantics. The mapping rules of this level are shown in Figure 7.
4. **Clause-Level**: Representing the highest level of abstraction, this level concentrates on the principal clauses of the SQL query, masking all details within those clauses. Operators like "**WHERE**" and "**<IUE>**" are kept for the clause level semantics.

Previous studies, such as DAIL-SQL [8], mainly focus on the Keywords-Level similarity. However, they typically overlook the keyword order because they rely on Jaccard Similarity calculations. In contrast, PURPLE models logical composition through a four-level automaton, representing keyword selection and ordering. This method of demonstration selection via the automaton framework facilitates the selection of essential composition knowledge by LLMs, which is not addressed by previous research.

For instance, DAIL-SQL [8] considers a skeleton like:

```
SELECT _ FROM _ JOIN _ ON _ = _ WHERE _ = _
EXCEPT
SELECT _ FROM _
```

as same with the skeleton shown in Section II-C. However, it failed to provide accurate composition knowledge for LLMs. Conversely, PURPLE prioritizes demonstrations as exemplified in Figure 2, as these can be matched through the Structure-Level automaton. The automaton design enhances the ability of PURPLE to select more relevant compositional knowledge for LLMs, improving overall effectiveness in SQL writing.

2) Automaton Construction: The automaton is constructed by parsing SQL skeletons extracted from all of the demonstrations $\mathcal{E}$. For each demonstration e_i as shown in Section III-A, we mask the database-specific tokens in the target SQL $\mathcal{Y}^{e_i}$ to get the skeleton $\mathcal{S}^{e_i}$. We parse all skeletons into basic elements, that we use to construct the Detail-Level automaton.

In addition, we add two specialized state nodes, denoted as "**<START>**" and "**<END>**", which serve as the initial and terminal states respectively. As the level of abstraction increases, more details are progressively masked.

We build the automaton for demonstration selection. To accelerate the selection process, we store the index of each demonstration within the "**<END>**" state node of its corresponding automaton. As we process the predicted skeleton through to the "**<END>**" state, the stored index helps to retrieve all demonstrations sharing identical automaton states. An *empty list* will be returned if a state sequence is absent in the demonstrations. Furthermore, we will remove all of the out-of-vocabulary tokens before parsing the predicted skeletons, which are introduced by the skeleton prediction model.

3) Automaton Matching: Our approach takes only identical sequences of automaton states as matches. This approach simplifies the extraction of demonstrations that align with each predicted skeleton across four levels of abstraction. Leveraging both the top-predicted skeletons and multiple abstraction levels, selecting demonstrations is a non-trivial task.

Algorithm 1: Demonstration Selection Algorithm

```
   Inputs : Automaton list $\mathcal{A}$; Query instance $\mathcal{Q}$
   Output: Selected demonstrations $\mathcal{E}'$
1  Procedure DEMONSTRATION-Selection($\mathcal{A}$, $\mathcal{Q}$):
2      $\mathcal{I}, \mathcal{E}' \leftarrow [\,]; p \leftarrow p_0$
3      for each $i \in [1, ..., 4]$ do
4          for each $j \in [1, ..., k]$ do
               // Get index by automaton
5              $\mathcal{I}$.append(MATCH($\mathcal{A}[i]$, $\mathcal{Q}.pred[j]$))
6      while NOT-Empty($\mathcal{I}$) do
7          for each $a \in$ GET-Top($\mathcal{I}$, $p$) do
               // Select demonstrations
8              $\mathcal{E}'$.append(POP-Demo($a$, $\mathcal{E}'$))
           // Higher generalization ability
9          $p \leftarrow$ INCREASE-Generalization($p$)
10     return $\mathcal{E}'$
```

PURPLE gives preference to skeletons that have high probability according to the model predictions and correspond to matches at lower levels of abstraction. This is based on the understanding that a higher probability prediction coupled with a lower abstraction level typically indicates a more precise match. Conversely, lower predicted probabilities and matches at higher levels of abstraction indicate greater generalization capacity but introduce more noise. PURPLE tries to balance the robustness and efficiency of the selection process.

The demonstration selection algorithm is shown in Algorithm 1, which prioritizes the selection of demonstrations with a higher prediction probability and a lower level of abstraction. The input $\mathcal{A}$ is the constructed automaton as shown in Section IV-C2, and $\mathcal{A}[i]$ represents the automaton at abstraction level i. The query instance $\mathcal{Q}$ stores the predicted skeletons as $\mathcal{Q}.pred$, and $\mathcal{Q}.pred[j]$ represents the j-th skeleton. The MATCH function (line 5) identifies indices of demonstrations that align with the j-th skeleton at abstraction level i. The preferential matching sequence $\mathcal{I}$ is a list with a size of $4 * k$, which stores matched indices (lines 2-5). The parameter p, for balancing precision and generalization, starts at p_0 and is adjusted by INCREASE-GENERALIZATION. As p increases, more demonstrations are considered (line 7). The GET-TOP function selects the top-p indices, while POP-DEMO retrieves matching demonstrations, ensuring compatibility across abstraction levels and avoiding duplicates in $\mathcal{E}'$.

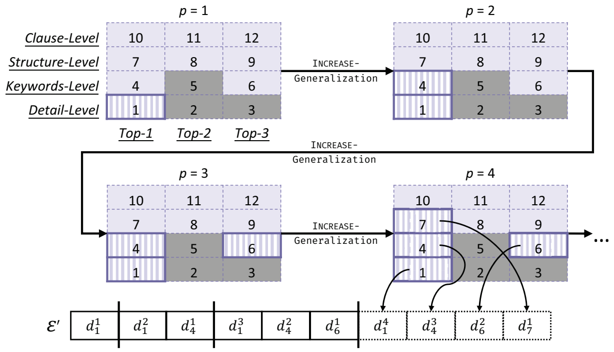


Fig. 8: Demonstration selection example.

Taking Figure 8 as an example, we represent the $\mathcal{I}$ as a matrix and discuss the detailed selection process (lines 6-9). In this matrix, columns correspond to the top-k (with $k = 3$ in our example) predicted skeletons, and rows correspond to four abstraction levels. Gray cells within the matrix indicate the absence of matching demonstrations for that specific combination. For instance, cell 2 means missing the Detail-Level match for the second skeleton. We start with $p = 1$ and increase it by 1 at each iteration. We select the top-p matches at each step, highlighted by purple strips in the figure. Each demonstration added to the selected demonstration queue $\mathcal{E}'$ is represented as d_i^j, meaning the j-th demonstration from the i-th cell. For example, in the first step with $p = 1$, demonstration d_1^1 is added to $\mathcal{E}'$, and in the second step with $p = 2$, d_1^2 and d_4^1 are added, as cells 2 and 3 lack matches. This process continues until no further demonstrations are contained in $\mathcal{I}$.

The value of p_0 and the INCREASE-Generalization function could be guided by the size of the automaton. A smaller automaton size suggests a higher density of demonstrations within each automaton state, which may introduce greater noise into the selection process. For instance, in our analysis of the Spider benchmark, we analyze the distribution of "**<END>**" states and their respective distribution within the four levels of automaton abstraction, finding proportions of $912 : 708 : 363 : 59$. Consequently, we set p_0 to 1, with p increasing by 1 at every step, aiming for a simplified expected matching ratio of $4 : 3 : 2 : 1$ across the abstraction levels, balancing precision and generalization. The remaining demonstrations are chosen randomly to fully utilize the budget.

Automaton with four-level abstraction can model logical operator composition knowledge across varying granularities, which is advanced in augmenting both the generalization and fuzzification capacities for the demonstration selection. We acknowledge that matching demonstrations at a higher level of abstraction can encompass broader logic by masking finer details, which could introduce uncertainty into the selection process. However, this broader perspective is crucial because it retains the fundamental knowledge of logical operator compositions, and the introduced minor errors could be identified and fixed by LLMs. Through this approach, PURPLE could extract demonstrations that contain the requisite operator composition knowledge, thereby enhancing its performance.

D. Database Adaption

The hallucination problem of existing LLMs results in the generation of invalid SQL during NL2SQL translation. Especially, SQL is related to the database schema and DBMS. Such a problem will cause a performance decline and lead to inconsistent translations. Through a detailed analysis of the LLMs outputs, we categorize common errors and develop algorithms to adapt the generated SQL to specific database schema and SQL dialect. We also incorporate an execution-consistency strategy to stabilize the translation outputs.

1) SQL Adaption: Modern LLMs benefit from extensive pre-training corpora, which equips the model with basic SQL knowledge. However, the corpora contain SQL variations from multiple DBMSs, which can result in translations that include syntax not supported by current DBMS.

TABLE 2: Error types and corresponding example

Error Type	Example

Table-Column-Mismatch	SELECT **T2.title** FROM cartoon AS T1 JOIN
	tv_channel AS T2 ON T1.channel = T2.id
	WHERE T2.series_name = "Sky Radio";

Column-Ambiguity	SELECT **maker**, model FROM car_makers
	JOIN model_list ON car_makers.id =
	model_list.maker JOIN car_names
	ON model_list.model = car_names.makeid;

Missing-Table	SELECT COUNT(DISTINCT language)
	FROM countrylanguage
	WHERE isofficial = 'T' AND **indepyear** < 1930;

Function-Hallucinations	SELECT **CONCAT(first_name, ' ', last_name)**
	AS full_name FROM players
	ORDER BY birth_date;

Schema-Hallucinations	SELECT **T1.course_id**, COUNT(*)
	AS count FROM transcript_contents AS T1
	JOIN student_enrolment_courses AS T2 ON
	T1.student_course_id = T2.student_course_id
	JOIN transcripts AS T3 ON
	T1.transcript_id = T3.transcript_id
	GROUP BY T1.course_id ORDER BY count DESC;

Aggregation-Hallucinations	SELECT **COUNT(DISTINCT series_name, content)**
	FROM tv_channel;

We identify six primary error categories, as shown in Table 2, which provides examples of invalid SQL for each category. We design heuristic algorithms for each error type. The algorithms fix the SQL queries that result in execution errors. So that PURPLE ensures that the SQL adaption strategy does not introduce undesired side effects to the valid SQL.

- **Table-Column Mismatch:** LLMs reason based on statistical patterns, leading to the incorrect alignment of columns to tables. As illustrated in Table 2, the column *title* belongs to the table *cartoon*, rendering *T2.title* an error. We rectify such errors by mapping the column to its correct table and adjusting the table identifier accordingly.
- **Column-Ambiguity:** A SQL might be invalid if multiple tables contain a column with the same name ambiguity. We randomly assign the column to one of its potential tables, ensuring its unique identification.
- **Missing-Table:** As denoted in Table 2, the column *indepyear* belongs to the table *country*, which is absent in the SQL. We fix this by including the table into the FROM clause based on primary-foreign key relationships.
- **Function-Hallucinations:** Certain functions like *CONCAT* are not supported in SQLite, resulting in invalid SQL. Our immediate solution is to omit the unsupported function call. An optimal solution would involve mapping functions across different DBMSs for future work.
- **Schema-Hallucinations:** LLMs may generate SQL referencing non-existent tables or columns within a given schema. For instance, Table 2 highlights that the column *course_id* is not present in any of the tables. We tackle this by identifying and substituting it with a column having a minimal string edit distance.
- **Aggregation-Hallucinations:** Aggregation functions in SQLite are designed to take a single column as input. To rectify errors like the one in Table 2, we divide the *COUNT* function into two separate counts, preserving the *DISTINCT* keyword for both columns.

PURPLE offers solutions for the six most common LLM-induced SQL errors. In our implementation, we attempt to rectify a non-executable SQL up to five times.

2) Consistency Strategy: Existing works like SQL-PaLM [3], C3 [11], and DAIL-SQL [8] integrate the execution-consistency strategy in stabilizing LLMs-based NL2SQL translations. We integrate this strategy into PURPLE with an increase in the cost of output tokens.

In detail, PURPLE prompts the LLMs to produce n SQL translations for every API call. SQL adaption process will be executed for the generated invalid SQL. Subsequently, each executable SQL is executed against the database. PURPLE then employs a voting mechanism based on the SQL execution results. The first SQL that yields the consensus execution result is selected as the output.

The hallucination of LLMs is an unavoidable issue. A categorization of issues stemming from hallucinations is beneficial in fixing those bugs. The fixing process is safe because it does not have side effects on the executable SQL. The database adaptation module utilizes database insights and DBMS characteristics to efficiently rectify erroneous SQL queries.

V. Experiments

In this section, we evaluate the overall performance of PURPLE. We discuss the trade-off between performance and API cost. Furthermore, we explore the robustness of PURPLE with various hyper-parameters and LLMs. Additionally, we conduct ablation studies on each module.

A. Experimental Setup

1) Benchmarks: We evaluate PURPLE on four popular NL2SQL benchmarks: Spider [10], Spider-DK [28], Spider-SYN [29], and Spider-Realistic [30]. The statistics about these benchmarks can be found in Table 3.

Spider is a popular benchmark for NL2SQL translation, consisting of 200 databases with multiple tables and 10,181

TABLE 3: The statistics of NL2SQL benchmarks

Benchmark	Queries	Databases	Average length of NL queries	Average length of target SQL
Spider(Train)	8,659	146	66.6	122.9
Spider(Validation)	1,034	20	68.0	106.7
Spider-DK	535	10	66.0	109.5
Spider-Realistic	508	20	64.8	115.3
Spider-SYN	1,034	20	68.8	106.7

NL-to-SQL pairs. It demands a comprehensive understanding of multi-table database relations, targeting performance evaluation on complex SQL translation based on unfamiliar domains. We evaluate PURPLE on the validation set of Spider, and we take the training set as the demonstration.

Spider-DK is a more challenging version of the Spider validation set. Such a benchmark requires the NL2SQL strategy to know about domain-specific knowledge for the SQL generation. Preliminary observations indicate many approaches struggle with this heightened domain-specific demand.

Spider-Realistic emphasizes the challenges of text-table alignments. It provides a more realistic scenario by omitting explicit mentions of column names and requires approaches to map NL terms to relevant database schema items adeptly.

Spider-SYN stems from the Spider. It modifies NL queries by swapping schema-related terms with handpicked synonyms, challenging the reliance on lexical matching.

2) Evaluation Metrics: We employ three evaluation metrics to assess the performance of PURPLE comprehensively: *Exact-Set Match (EM)* accuracy, *Execution Match (EX)* accuracy [10], and *Test-Suite (TS)* accuracy [31]. The detailed description of the metrics is as follows.

EM accuracy is one of the official evaluation metrics of Spider, which uses a set comparison for each clause. While precise, EM might yield false negatives due to new syntax structures from semantic parsers.

EX accuracy is also officially supported by Spider, which checks the congruence of executed predicted SQL query results with expected outcomes. EX can sometimes return false positives when differing SQL queries yield identical results but potentially with varied semantics.

TS accuracy aims to rectify the EX metric by employing a distilled test suite of databases [31]. The distilled database is created by selecting a small subset from numerous random databases that can distinguish between correct and nearly correct queries, ensuring high code coverage. We follow the original script[4] to generate an augmented 100-fold distilled database for evaluation.

3) Baselines: Existing NL2SQL approaches are used for comparison to show the performance of PURPLE. We choose some SOTA LLMs-based approaches for comparison, including ChatGPT-SQL [5], C3 [11], DIN-SQL [2] and DAIL-SQL [8]. Basic few-shot strategies are shown in DIN-SQL [2], and we also include the GPT4 results for comparison. We also report the performance of some PLMs-based approaches on Spider for reference, including PICARD [21], RASAT [32], RESDSQL [22] and Graphix-T5 [33].

ChatGPT-SQL aims to thoroughly assess the zero-shot NL2SQL capabilities of ChatGPT. The predictions from this approach have been made publicly available, and we leverage these open-source results for our comparative analysis.

C3 is a zero-shot LLMs-based approach by hand-crafted instruction. C3 also proposes to reduce the input length of LLM API calls but fails to control the output length.

DIN-SQL employs a few-shot approach and has achieved leading performance in terms of EX on the Spider. DIN-SQL incorporates CoT for performance enhancement. Additionally, DIN-SQL reports the result of **GPT4 few-shot** and **GPT4 zero-shot** approaches, which we include in our comparisons.

DAIL-SQL implement demonstration selection by analyzing NL query and SQL similarity. This adaptable demonstration selection strategy has shown promising results, especially when integrated with the capabilities of GPT4.

PICARD, **RASAT**, **RESDSQL**, and **Graphix-T5** represent SOTA PLMs-based methods. They are all based on the T5 model with improving the encoder, decoder, or task formulation. We report their optimal performance for comparison.

4) Implementation Details: We employ ChatGPT (gpt-3.5-turbo-0613)[5] and GPT4 (gpt-4-0613)[6] for the SQL generation. Our training environment operates on Centos 7.9, with a 64-core CPU, 512GB of memory, and 8 NVIDIA A100 GPUs. For the schema pruning module, we set $\tau_p = 0.5$, $\tau_n = 5$. We fine-tune a T5-3B model for skeleton prediction, selecting the top-3 skeletons. The automaton matching hyper-parameters followed the setting shown in Section IV-C3. For the cost saving, comparisons involving GPT4-based approaches are confined to Section V-B and Section V-F. All other experimental evaluations are conducted using ChatGPT.

B. Overall Performance

We evaluate PURPLE by comparing it against SOTA LLMs-based and PLMs-based approaches for a comprehensive view.

TABLE 4: Translation accuracy on Spider.

Strategy	EM%	EX%	TS%
PICARD	75.5	79.3	69.4
RASAT	75.3	80.5	70.3
RESDSQL	**80.5**	84.1	73.5
Graphix-T5	77.1	81.0	74.9
ChatGPT-SQL (ChatGPT)	37.9	70.1	60.1
C3 (ChatGPT)	43.1	81.8	72.1
Zero-shot (GPT4)	42.4	72.9	64.9
Few-shot (GPT4)	54.3	76.8	67.4
DIN-SQL (GPT4)	60.1	82.8	74.2
DAIL-SQL (GPT4)	68.7	83.6	76.2
PURPLE (ChatGPT)	76.1	84.8	80.1
PURPLE (GPT4)	**80.5**	**87.8**	**83.3**

Table 4 illustrates that when augmented with GPT4, PURPLE surpasses other LLMs-based strategies across all metrics on the validation set of Spider, including EM, EX, and TS. Remarkably, PURPLE remains superior even when coupled with the comparatively weak ChatGPT. DAIL-SQL achieves

[4] https://github.com/ruiqi-zhong/TestSuiteEval

[5] https://openai.com/chatgpt

[6] https://openai.com/gpt-4

an 83.6% EX among its LLMs-based counterparts but only 76.2% on TS. PURPLE with GPT4 enhances the performance by a large margin, which means a 4.2% improvement on EX and a 7.1% improvement on TS than DAIL-SQL. A typical challenge for existing LLMs-based NL2SQL approaches is their low EM because of their inability to guide the generative process of LLMs. However, PURPLE achieves an 11.8% improvement over DAIL-SQL and a 20.4% improvement over DIN-SQL in EM, showing the reliability of PURPLE.

In addition, we compare PURPLE with SOTA PLMs-based approaches on the Spider. While PURPLE incorporates a fine-tuning phase, its primary application is demonstration retrieval to enhance the LLMs. PURPLE reaches the top EM score compared with all of the PLMs-based approaches, in which PURPLE achieves 80.5% EM on the spider validation set. Achieving the highest score in EM, EX, and TS, PURPLE shows the ability of LLMs-based NL2SQL approaches to outperform their PLMs-based counterparts.

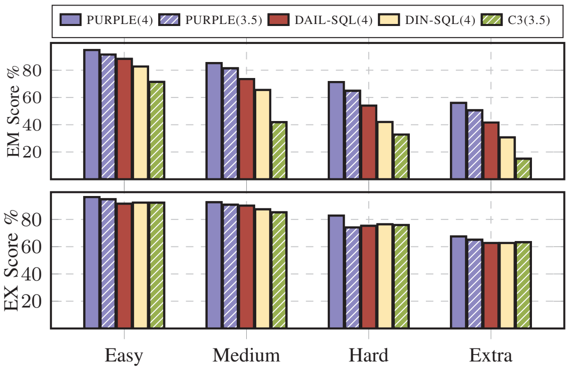


Fig. 9: Comparison of the EM/EX score on the Spider validation set regarding SQL hardness levels.

Figure 9 shows the performance of various approaches based on SQL hardness levels for the Spider validation set. We follow the official evaluation scripts for the hardness classification. The legend shows the name of approaches and the LLMs, such as *PURPLE(4)* means PURPLE with GPT4, *C3(3.5)* represents C3 with ChatGPT. When augmented with GPT4, PURPLE consistently achieves the highest performance across all SQL hardness levels. Notably, even with the relatively weak ChatGPT, PURPLE still surpasses other approaches in terms of EM, regardless of SQL hardness.

An observation is that PURPLE advances in handling the *extra hard* SQL translations. This ability can be attributed to its emphasis on operator composition knowledge, thereby enhancing the LLM with complex SQL generation capacity. Conversely, DIN-SQL employs CoT to facilitate LLMs in managing complex SQL constructions. While these CoT demonstrations help LLMs understand user intention, they fail to include SQL operator composition knowledge. In addition, C3 focuses on syntactic constraints within its designed instruction. However, such hand-crafted instructions are insufficient to provide the necessary operator composition knowledge. DAIL-SQL utilizes both NL and SQL similarity for demonstration selection, but the similarity function can not capture the operator composition similarity between two SQL queries as described in Section IV-C2, thereby failing to address the limitations of LLMs. PURPLE selects the demonstrations based on the logical operator composition, which successfully improves the performance of existing general LLMs.

C. Generalization Ability

Generalization ability is a vital aspect when evaluating NL2SQL approaches. An NL2SQL system will likely be deployed on databases unseen during training. We utilize Spider-DK, Spider-SYN, and Spider-Realistic benchmarks to evaluate the generalization ability. We train PURPLE on the Spider dataset and test its EM and EX accuracy on the three benchmarks. We compare the performance of two other ChatGPT-based strategies, ChatGPT-SQL and C3.

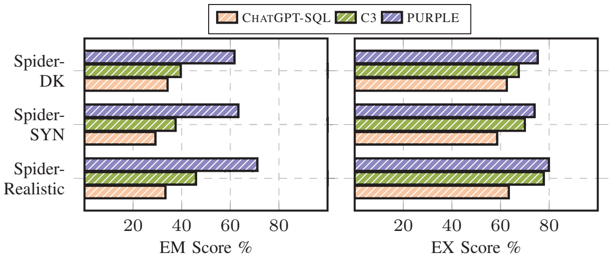


Fig. 10: Comparison of EM/EX scores on Spider-DK, Spider-SYN and Spider-Realistic.

As shown in Figure 10, PURPLE achieves the best EM score, a notable achievement for LLM-based approaches that often struggle in this area. Specifically, PURPLE registers EM scores of 61.7%, 63.3%, and 71.1% on Spider-DK, Spider-SYN, and Spider-Realistic benchmarks, respectively. These results are over 22% better than C3, demonstrating a superior ability to generate accurate SQL compared to other methods.

Figure 10 also shows that PURPLE consistently maintains high EX scores across the three benchmarks, with 75.3%, 74.0%, and 79.9% on Spider-DK, Spider-SYN, and Spider-Realistic, respectively. This uniformity in performance illustrates the robustness of PURPLE relative to its counterparts.

Although PURPLE incorporates fine-tuning for demonstration retrieval, it avoids a performance drop across varying data distributions. Because the fine-tuned model is utilized to enhance the operator composition knowledge as the intermediary.

D. Cost v.s. Performance

PURPLE forms the prompt based on the token number to control the budget for each SQL translation. We evaluate the performance of PURPLE on the Spider under varying budget constraints, focusing on input length and the number of responses. We evaluate with input token limitation (len) of $512, 1024, 2048, 3072$ and consistency numbers (num) from $1, 10, 20, 30, 40$. The num is to control the generated token number. Figure 11 shows the accuracy under different budgets.

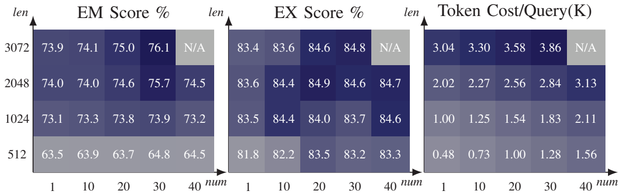


Fig. 11: PURPLE (ChatGPT) performance and token consumption under various budget settings. len represents prompt length, num represents consistency number.

The performance of PURPLE tends to enhance with an increased budget. However, the increase becomes marginal when the input length surpasses 2,048 tokens. This is because adding more tokens offers diminishing returns on its ability to generalize. Due to LLM limitations, a single call can process only up to 4,096 tokens, shown as N/A in Figure 11.

Our default configuration for PURPLE is set with an input length of 3,072 and a consistency number of 30. For context, DIN-SQL with GPT4 consumes roughly 10,000 tokens for each query translation. C3 uses about 8,000 tokens, splitting between 1,000 for input and 7,000 for output. DAIL-SQL, which can adjust input and output length, works best with around 3,000 tokens. Meanwhile, PURPLE outperforms these with only 1,250 tokens in ChatGPT, highlighting its efficiency.

E. Robustness of Demonstration Selection

We evaluate the robustness of our demonstration selection algorithm by varying the initial parameter p_0 and adjusting the INCREASE-Generalization method, as shown in Algorithm 1. We also explore how inaccuracies in skeleton prediction impact performance.

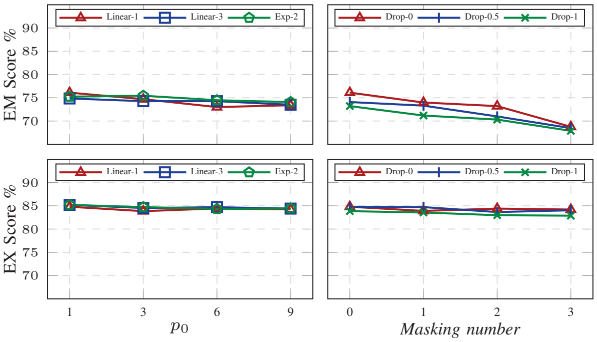


Fig. 12: Robustness evaluation for demonstration selection

The left side of Figure 12 shows the performance of PURPLE with various p_0 and generalization methods. For example, *Linear-1(3)* adds 1(3) to p at each step, and *Exp-2* doubles p at each step. We found that the changes in performance are minor, less than 3% for EM and less than 1.5% for EX. This indicates that the performance of PURPLE is relatively stable.

On the right side of Figure 12, we evaluate the effects of inaccurate skeletons on PURPLE. To simulate inaccuracies, we introduced two types of noise: $masking\ number = x$ simulates missing detailed information by ignoring the first x layers of automaton abstraction, and $Drop-y$ randomly drops one predicted skeleton with a probability of y. For instance, $Drop - 0.5$ and $masking\ number = 2$ drops one predicted skeleton half the time and ignores the first two abstraction levels (Detail-Level and Keywords-Level) during demonstration selection. We observed a drop in EM scores with more noise. However, even in tough scenarios like only using Clause-Level information, PURPLE still achieves competitive EM scores, demonstrating its resilience to prediction inaccuracies.

F. Performance with Various LLMs

For the LLMs-based approaches, the selection of the specific LLM can lead to variations in performance. We evaluate the performance variations on the Spider of DIN-SQL, C3, DAIL-SQL, and PURPLE when utilizing either ChatGPT or GPT4. The results are shown in Table 5.

TABLE 5: EM/EX comparison between ChatGPT and GPT4.

Strategy	LLM	EM%	EX%
DIN-SQL	GPT4	60.1	82.8
	ChatGPT	43.0(-17.1)	75.5(-7.3)
C3	GPT4	50.7	82.1
	ChatGPT	43.1(-7.6)	81.8(-0.3)
DAIL-SQL	GPT4	68.7	83.6
	ChatGPT	65.1(-3.6)	81.3(-2.3)
PURPLE	GPT4	80.5	87.8
	ChatGPT	76.1(-4.4)	84.8(-3.0)

PURPLE consistently outperforms others, whether using ChatGPT or GPT4. Meanwhile, DIN-SQL exhibits a sensitivity to the LLMs. DIN-SQL employs CoT methodology, which relies on the reasoning capabilities inherent in GPT4. ChatGPT struggles with complex reasoning tasks, thereby increasing the risk of error propagation [34]. The sensitivity to the LLMs not only raises its cost but also undermines its robustness.

C3 shows stable performance across both LLMs. However, it underutilizes the capabilities of GPT4. The hand-crafted instructions limit its SQL knowledge. Lacking operator composition knowledge in prompt restricts the potential enhancements.

DAIL-SQL exhibits a parallel trend in performance variability between GPT4 and ChatGPT, similar to PURPLE. Because they both propose to utilize an adaptable demonstration selection strategy. However, DAIL-SQL lacks sufficient operator composition knowledge to achieve better performance.

PURPLE consistently outperforms other strategies with both ChatGPT and GPT4. It stays accurate in tight resource settings, showing the importance of giving LLMs the logical operator composition knowledge they need for tasks.

G. Ablation Study

We conduct an ablation study to show the contribution of each module in PURPLE. The results are shown in Table 6.

The schema pruning module simplifies tasks, helping LLMs focus on key information for SQL generation. The EM and EX scores suffer from a large drop without such a module (-Schema Pruning). To evaluate the effectiveness of the Steiner Tree-based pruning strategy, we compared it with the pruning

TABLE 6: Ablation Study.

Strategy	EM%	EX%
PURPLE (ChatGPT)	76.1	84.8
-Schema Pruning	71.2(-4.9)	83.4(-1.4)
-Steiner Tree	75.0(-1.1)	84.4(-0.3)
-Demonstration Selection	59.1(-17.0)	81.6(-3.2)
-Database Adaption	74.7(-1.4)	81.8(-3.0)
+Oracle Skeleton	78.8(+2.7)	86.8(+2.0)

approach used by RESDSQL (-Steiner Tree). Both the EM and EX scores are lower when using the RESDSQL method. This is because it requires LLMs to process more information, highlighting the efficiency of our approach.

Demonstration selection is the key module of PURPLE. Randomly selecting demonstrations (-Demonstration Selection) greatly reduced EM scores, showing the importance of composition knowledge for the SQL generation of LLMs.

The database adaptation module further builds upon the successes of the previous modules, contributing to the stabilization of model outputs and mitigating hallucination issues.

Additionally, we conducted an oracle-setting experiment. We replace the predicted skeletons with the oracle skeleton. PURPLE with ChatGPT (+Oracle Skeleton) achieves an EM score of 78.8% and an EX score of 86.8%. This result highlights the importance of accurately predicting logical composition knowledge in the overall performance of PURPLE. Improving skeleton prediction could further boost results.

VI. Related Works

The NL2SQL task has been under investigation for decades with the advancements of NLP. Early studies like [35]–[42] on NL2SQL mainly focus on rule-based mapping, which has limited generalization ability. Modern NL2SQL approaches incorporate SOTA models to optimize performance. We classify the LMs-based NL2SQL approaches into PLMs-based and LLMs-based NL2SQL as shown in Section II.

PLMs-based NL2SQL. Fine-tuning PLMs as part of a sequence-to-sequence paradigm is one of the popular approaches for NL2SQL. Such fine-tuning techniques empower the development of custom modules on the foundation of PLMs. Works like RAT-SQL [43], GNN [44], GlobalGNN [45] BIRDGE [19], LGESQL [46], S^2SQL [47], RASAT [32] and Graphix-T5 [33] have introduced novel encoder architectures for enhanced semantic comprehension. IRNet [48], SmBoP [49], NatSQL [50], PICARD [21], CATSQL [51], and SC-Prompt [52] focus on reducing decoder complexity, while others such as RESDSQL [22], N-best [53], GAR [54], GenSQL [55] and MetaSQL [56] integrate ranking models to elevate performance. However, the PLMs-based approaches are constrained by model and pre-trained corpus sizes, leading to misunderstanding of user intentions. As models grow, their adaptability for NL2SQL tasks diminishes, prompting a shift of research attention towards LLMs-based methodologies.

LLMs-based NL2SQL. Techniques for integrating LLMs into NL2SQL can be categorized according to whether demonstrations are employed in the prompt, leading to categorizations as either zero-shot or few-shot methodologies. Zero-shot approaches, such as those explored in [5], [7], [11], [57], aim to refine translation precision through instruction design. Few-shot methodologies [1]–[3], [58]–[63], contains related knowledge for teaching the LLMs about how to handle the translation task on hand. For instance, DAIL-SQL [8], SYNCHROMESH [1] and Linyong et al. [59] prioritize SQL-aligned demonstrations, while SKILL-KNN [61] prefers to include semantically similar ones. Some propose retrieving demonstrations with the coverage for the prompt [63]. CoT is an LLMs-based technique that has been applied on NL2SQL [2], [58], [60], which identifies that variations in CoT style can influence performance outcomes. Several multi-turn approaches [61], [62] propose to refine the generated SQL in multi-turn interactions with LLMs, achieving higher accuracy while suffering from high API cost. A common shortcoming among existing LLMs-based approaches is their inability to achieve high EM scores, often attributed to the challenge of controlling the generation process. In response to the observed limitations of LLMs in SQL writing, PURPLE focuses on extracting essential logical operator composition knowledge for logical enhancement, leading to a new SOTA performance.

VII. Conclusion and Future Work

We introduced PURPLE, a novel LLMs-based NL2SQL approach that enhanced translation precision through demonstration selection. PURPLE models the operator composition knowledge by a four-level automaton, and related automaton construction and matching strategies are designed for demonstration selection. Schema pruning and skeleton prediction facilitate this selection process, and the database adaptation module serves to stabilize outputs and mitigate hallucination issues. PURPLE successfully enhanced the LLMs with SQL operator composition knowledge, achieving reliable performance on four popular benchmarks. We also evaluated the robustness and the influence of LLM selection for PURPLE.

One promising research direction is the development of generation-based prompting methods. While PURPLE effectively retrieves existing demonstrations to construct prompts, this retrieval-based strategy is inherently limited by the available pool of demonstrations. Involving generating prompts directly using PLMs is a potentially more flexible approach. This method could offer a more generalized and intuitive way to create prompts. However, the primary challenge with a generation-based approach lies in fine-tuning PLMs to produce optimized prompts effectively. Although there has been some success with using reinforcement learning for prompt optimization in prior research [64], [65], fine-tuning PLMs specifically for prompt generation presents difficulties. Using existing demonstrations as a basis, like PURPLE, could serve as a valuable starting point for developing more advanced generation-based prompting methods in the future.

VIII. Acknowledgments

The authors thank the reviewers for the valuable feedback. This work was supported by NSFC (Grant No. 62272106). The corresponding authors are Zhenying He and X.Sean Wang.

REFERENCES

[1] G. Poesia, A. Polozov, V. Le, A. Tiwari, G. Soares, C. Meek, and S. Gulwani, "Synchromesh: Reliable code generation from pre-trained language models," in *ICLR*, 2022.

[2] M. Pourreza and D. Rafiei, "DIN-SQL: decomposed in-context learning of text-to-sql with self-correction," *CoRR*, 2023.

[3] R. Sun, S. Ö. Arik, H. Nakhost, H. Dai, R. Sinha, P. Yin, and T. Pfister, "Sql-palm: Improved large language model adaptation for text-to-sql," *CoRR*, 2023.

[4] N. Rajkumar, R. Li, and D. Bahdanau, "Evaluating the text-to-sql capabilities of large language models," *CoRR*, 2022.

[5] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability," *CoRR*, 2023.

[6] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," *CoRR*, 2023.

[7] Z. Gu, J. Fan, N. Tang, S. Zhang, Y. Zhang, Z. Chen, L. Cao, G. Li, S. Madden, and X. Du, "Interleaving pre-trained language models and large language models for zero-shot NL2SQL generation," *CoRR*, 2023.

[8] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *CoRR*, 2023.

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *NeurIPS*, 2022.

[10] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," in *EMNLP*, 2018, 3911–3921.

[11] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, L. Chen, J. Lin, and D. Lou, "C3: zero-shot text-to-sql with chatgpt," *CoRR*, 2023.

[12] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *CoRR*, 2023.

[13] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *NeurIPS*, 2020.

[14] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019, 4171–4186.

[15] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *CoRR*, 2019.

[16] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, 140:1–140:67, 2020.

[17] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, 2020.

[18] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," *CoRR*, 2022.

[19] X. V. Lin, R. Socher, and C. Xiong, "Bridging textual and tabular data for cross-domain text-to-sql semantic parsing," in *EMNLP*, 2020, 4870–4888.

[20] C. Zhou, J. He, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, "Prompt consistency for zero-shot task generalization," in *EMNLP*, 2022, 2613–2626.

[21] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: parsing incrementally for constrained auto-regressive decoding from language models," in *EMNLP*, 2021, 9895–9901.

[22] H. Li, J. Zhang, C. Li, and H. Chen, "RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql," in *AAAI*, 2023, 13 067–13 075.

[23] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, 2017, 2999–3007.

[24] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, vol. 22, no. 1, 55–89, 1992.

[25] V. Hristidis and Y. Papakonstantinou, "DISCOVER: keyword search in relational databases," in *VLDB*, 2002, 670–681.

[26] I. Ljubic, "Solving steiner trees: Recent advances, challenges, and perspectives," *Networks*, vol. 77, no. 2, 177–204, 2021.

[27] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," in *ACL*, T. Luong, A. Birch, G. Neubig, and A. M. Finch, Eds., 2017, 56–60.

[28] Y. Gan, X. Chen, and M. Purver, "Exploring underexplored limitations of cross-domain text-to-sql generalization," in *EMNLP*, 2021, 8926–8931.

[29] Y. Gan, X. Chen, Q. Huang, M. Purver, J. R. Woodward, J. Xie, and P. Huang, "Towards robustness of text-to-sql models against synonym substitution," in *ACL*, 2021, 2505–2515.

[30] X. Deng, A. H. Awadallah, C. Meek, O. Polozov, H. Sun, and M. Richardson, "Structure-grounded pretraining for text-to-sql," in *NAACL*, 2021, 1337–1350.

[31] R. Zhong, T. Yu, and D. Klein, "Semantic evaluation for text-to-sql with distilled test suites," in *EMNLP*, 2020, 396–411.

[32] J. Qi, J. Tang, Z. He, X. Wan, Y. Cheng, C. Zhou, X. Wang, Q. Zhang, and Z. Lin, "RASAT: integrating relational structures into pretrained seq2seq model for text-to-sql," in *EMNLP*, 2022, 3215–3229.

[33] J. Li, B. Hui, R. Cheng, B. Qin, C. Ma, N. Huo, F. Huang, W. Du, L. Si, and Y. Li, "Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing," in *AAAI*, 2023, 13 076–13 084.

[34] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *ICLR*, 2023.

[35] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *AAAI*, 1996, 1050–1055.

[36] A. Simitsis, G. Koutrika, and Y. E. Ioannidis, "Précis: from unstructured keywords as queries to structured databases as answers," *VLDBJ*, vol. 17, no. 1, 117–149, 2008.

[37] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *VLDB*, vol. 8, no. 1, 73–84, 2014.

[38] D. Saha, A. Floratou, K. Sankaranarayanan, U. F. Minhas, A. R. Mittal, and F. Özcan, "ATHENA: an ontology-driven system for natural language querying over relational data stores," *VLDB*, vol. 9, no. 12, 1209–1220, 2016.

[39] F. Li and H. V. Jagadish, "Nalir: an interactive natural language interface for querying relational databases," in *SIGMOD*, 2014, 709–712.

[40] J. Sen, C. Lei, A. Quamar, F. Özcan, V. Efthymiou, A. Dalmia, G. Stager, A. R. Mittal, D. Saha, and K. Sankaranarayanan, "ATHENA++: natural language querying for complex nested SQL queries," *VLDB*, vol. 13, no. 11, 2747–2759, 2020.

[41] H. Kim, B. So, W. Han, and H. Lee, "Natural language to SQL: where are we today?" *VLDB*, vol. 13, no. 10, 1737–1750, 2020.

[42] O. Gkini, T. Belmpas, G. Koutrika, and Y. E. Ioannidis, "An in-depth benchmarking of text-to-sql systems," in *SIGMOD*, 2021, 632–644.

[43] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers," in *ACL*, 2020, 7567–7578.

[44] B. Bogin, J. Berant, and M. Gardner, "Representing schema structure with graph neural networks for text-to-sql parsing," in *ACL*, 2019, 4560–4565.

[45] B. Bogin, M. Gardner, and J. Berant, "Global reasoning over database structures for text-to-sql parsing," in *EMNLP*, 2019, 3657–3662.

[46] R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, and K. Yu, "LGESQL: line graph enhanced text-to-sql model with mixed local and non-local relations," in *ACL*, 2021, 2541–2555.

[47] B. Hui, R. Geng, L. Wang, B. Qin, Y. Li, B. Li, J. Sun, and Y. Li, "S^2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers," in *ACL*, 2022, 1254–1262.

[48] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J. Lou, T. Liu, and D. Zhang, "Towards complex text-to-sql in cross-domain database with intermediate representation," in *ACL*, 2019, 4524–4535.

[49] O. Rubin and J. Berant, "Smbop: Semi-autoregressive bottom-up semantic parsing," in *ACL*, 2021, 12–21.

[50] Y. Gan, X. Chen, J. Xie, M. Purver, J. R. Woodward, J. H. Drake, and Q. Zhang, "Natural SQL: making SQL easier to infer from natural language specifications," in *EMNLP*, 2021, 2030–2042.

[51] H. Fu, C. Liu, B. Wu, F. Li, J. Tan, and J. Sun, "Catsql: Towards real world natural language to SQL applications," *VLDB*, vol. 16, no. 6, 1534–1547, 2023.

[52] Z. Gu, J. Fan, N. Tang, L. Cao, B. Jia, S. Madden, and X. Du, "Few-shot text-to-sql translation using structure and content prompt learning," *SIGMOD*, vol. 1, no. 2, 147:1–147:28, 2023.

[53] L. Zeng, S. H. K. Parthasarathi, and D. Hakkani-Tur, "N-best hypotheses reranking for text-to-sql systems," in *SLT*, 2022, 663–670.

[54] Y. Fan, Z. He, T. Ren, D. Guo, L. Chen, R. Zhu, G. Chen, Y. Jing, K. Zhang, and X. S. Wang, "Gar: A generate-and-rank approach for natural language to SQL translation," in *ICDE*, 2023, 110–122.

[55] Y. Fan, T. Ren, Z. He, X. S. Wang, Y. Zhang, and X. Li, "Gensql: A generative natural language interface to database systems," in *ICDE*, 2023, 3603–3606.

[56] Y. Fan, Z. He, T. Ren, C. Huang, Y. Jing, K. Zhang, and X. S. Wang, "Metasql: A generate-then-rank framework for natural language to sql translation," *CoRR*, 2024.

[57] S. Chang and E. Fosler-Lussier, "How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings," *CoRR*, 2023.

[58] X. Liu and Z. Tan, "Divide and prompt: Chain of thought prompting for text-to-sql," *CoRR*, 2023.

[59] L. Nan, Y. Zhao, W. Zou, N. Ri, J. Tae, E. Zhang, A. Cohan, and D. Radev, "Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies," *CoRR*, 2023.

[60] C. Tai, Z. Chen, T. Zhang, X. Deng, and H. Sun, "Exploring chain-of-thought style prompting for text-to-sql," *CoRR*, 2023.

[61] S. An, B. Zhou, Z. Lin, Q. Fu, B. Chen, N. Zheng, W. Chen, and J. Lou, "Skill-based few-shot selection for in-context learning," *CoRR*, 2023.

[62] C. Guo, Z. Tian, J. Tang, S. Li, Z. Wen, K. Wang, and T. Wang, "Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain," *CoRR*, 2023.

[63] A. Arora, S. Bhaisaheb, H. Nigam, M. S. Patwardhan, L. Vig, and G. Shroff, "Adapt and decompose: Efficient generalization of text-to-sql via domain adapted least-to-most prompting," *CoRR*, 2023.

[64] M. Deng, J. Wang, C. Hsieh, Y. Wang, H. Guo, T. Shu, M. Song, E. P. Xing, and Z. Hu, "Rlprompt: Optimizing discrete text prompts with reinforcement learning," in *EMNLP*, 2022, 3369–3391.

[65] P. Lu, L. Qiu, K. Chang, Y. N. Wu, S. Zhu, T. Rajpurohit, P. Clark, and A. Kalyan, "Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning," in *ICLR*, 2023.