A Survey of NL2SQL with Large Language Models: Where are we, and where are we going?

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo*,
Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang

NL2SQL Handbook: https://github.com/HKUSTDial/NL2SQL_Handbook

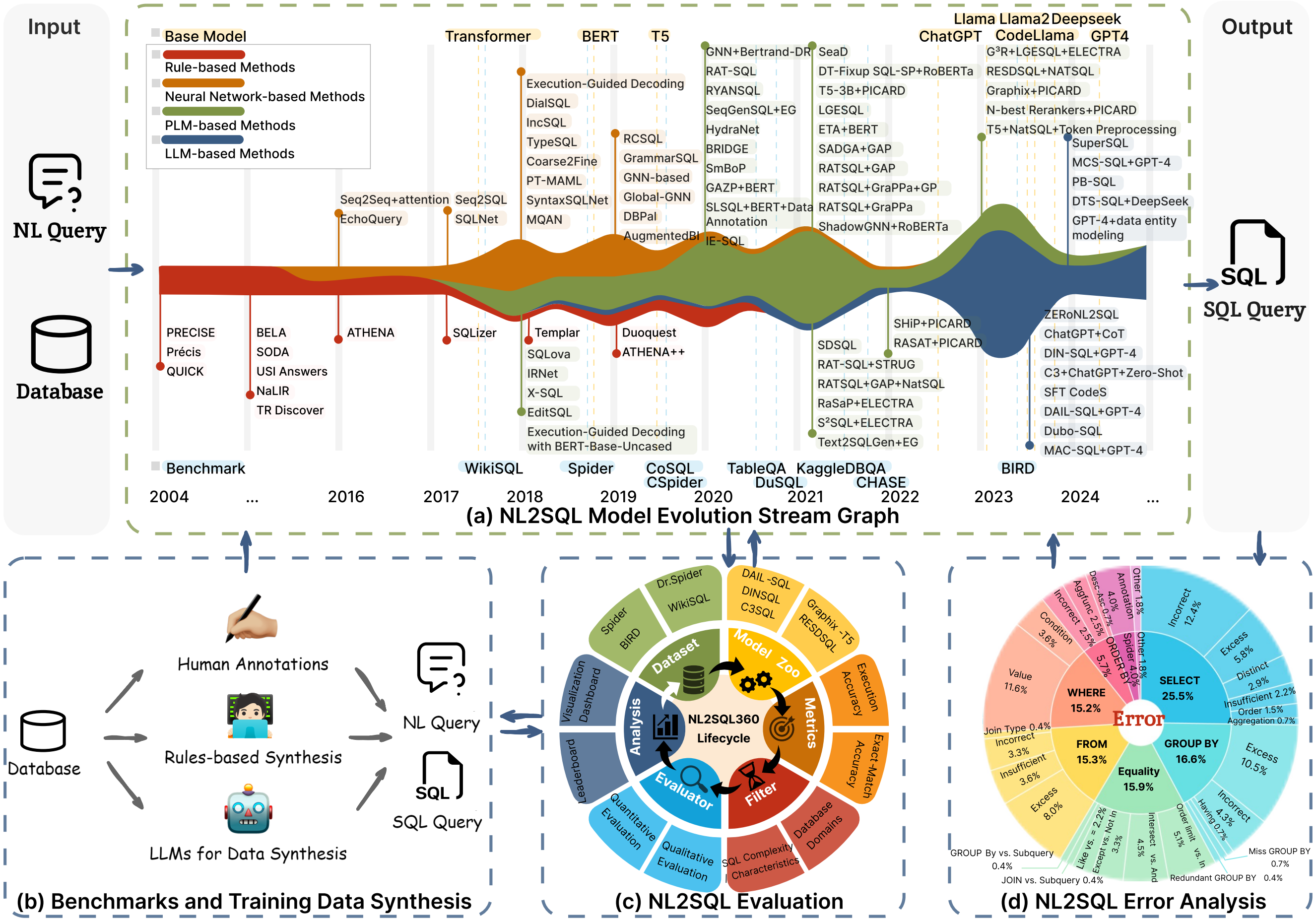


Fig. 1: An Overview of the Survey: The Lifecycle of the NL2SQL Task.

***Abstract*—Translating users' natural language queries (NL) into SQL queries (*i.e.*, NL2SQL) can significantly reduce barriers to accessing relational databases and support various commercial applications. The performance of NL2SQL has been greatly enhanced with the emergence of Large Language Models (LLMs). In this survey, we provide a comprehensive review of NL2SQL techniques powered by LLMs, covering its entire lifecycle from the following four aspects: (1) *Model:* NL2SQL translation techniques that tackle not only NL ambiguity and under-specification, but also properly map NL with database schema and instances; (2) *Data:* From the collection of training data, data synthesis due to training data scarcity, to NL2SQL benchmarks; (3) *Evaluation:* Evaluating NL2SQL methods from multiple angles using different metrics and granularities; and (4) *Error Analysis:* analyzing NL2SQL errors to find the root cause and guiding NL2SQL models to evolve. Moreover, we provide a rule of thumb for developing NL2SQL solutions. Finally, we discuss the research challenges and open problems of NL2SQL in the LLMs era.**

***Index Terms*—Natural Language to SQL, Database Interface, Large Language Models.**

I. Introduction

NATURAL Language to SQL (*i.e.*, NL2SQL), which converts a natural language query (NL) into an SQL query, is a key technique toward lowering the barrier to accessing relational databases [1]–[7]. This technique supports various important applications such as business intelligence, customer support, and more, making it a key step toward democratiz-

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Nan Tang and Yuyu Luo are with The Hong Kong University of Science and Technology (Guangzhou), China. E-mail: {xliu371, rjiang073, pma929}@connect.hkust-gz.edu.cn, {shuyushen, boyanli, yuyuluo, nantang}@hkust-gz.edu.cn.

Yuxin Zhang and Ju Fan are with Renmin University of China, Beijing, China. E-mail: {zhangyuxin159, fanj}@ruc.edu.cn.

Guoliang Li is with Tsinghua University, Beijing, China. E-mail: liguoliang@tsinghua.edu.cn.

*Corresponding author: Yuyu Luo (yuyuluo@hkust-gz.edu.cn).

ing data science [8]–[19]. Recent advancements in language models have significantly extended the frontiers of research and application in NL2SQL. Concurrently, the trend among database vendors to offer NL2SQL solutions has evolved from a mere notion to a necessary strategy [20], [21]. Therefore, it's important for us to understand the fundamentals, techniques, and challenges regarding NL2SQL.

In this survey, we will systematically review recent NL2SQL techniques through a new framework, as shown in Figure 1.

- **NL2SQL with Language Models.** We will first review existing NL2SQL solutions from the perspective of language models, categorizing them into four major categories (see Figure 1(a)). We will then focus on the recent advances in Pre-trained Language Models (PLMs) and Large Language Models (LLMs) for NL2SQL.
- **Benchmarks and Training Data Synthesis**. Undoubtedly, the performance of PLM- and LLM-based NL2SQL models is highly dependent on the amount and quality of the training data. Therefore, we will first summarize the characteristics of existing benchmarks and analyze their statistical information (*e.g.,* database and query complexity) in detail. We will then discuss methods for collecting and synthesizing high-quality training data, highlighting this as a research opportunity (see Figure 1(b)).
- **Evaluation.** Comprehensively evaluating NL2SQL models is crucial for optimizing and selecting models for different usage scenarios. We will discuss the multi-angle evaluation and scenario-based evaluation for the NL2SQL task (see Figure 1(c)). For example, performance can be assessed in specific contexts by filtering datasets based on SQL characteristics, NL variants, and database domains.
- **NL2SQL Error Analysis.** Error analysis is essential in NL2SQL research for identifying model limitations. We review existing error taxonomies, analyze their limitations, and propose principles for designing taxonomies for NL2SQL output errors. Using these principles, we create a two-level error taxonomy and utilize it to summarize and analyze NL2SQL output errors (see Figure 1(d)).

Next, we will introduce practical guidance for developing NL2SQL solutions, including a roadmap we designed for optimizing LLMs to NL2SQL task, along with a decision flow we created for selecting NL2SQL modules tailored to different NL2SQL scenarios. Finally, we will introduce some interesting and important open problems in the field of NL2SQL, including open-world NL2SQL tasks, cost-effective NL2SQL with LLMs, and trustworthy NL2SQL solutions.

Differences from Existing Surveys. Our survey distinguishes itself from existing NL2SQL surveys [22]–[26] and tutorials [27]–[29] in five aspects.

- We systematically review *the entire lifecycle of NL2SQL problem*, as shown in Figure 1. This lifecycle includes training data collection and synthesis methods (Figure 1(b)), various NL2SQL translation methodologies (Figure 1(a)), multi-angle and scenarios-based evaluations (Figure 1(c)), and NL2SQL output error analysis techniques (Figure 1(d)).
- We provide a more detailed and comprehensive summary of the inherent challenges in NL2SQL. Additionally, we analyze the technical challenges when developing a robust NL2SQL solution for real-world scenarios, which are often overlooked in other surveys.
- We particularly focus on recent advances in *LLM-based* NL2SQL methods, summarizing key modules and comparing different strategies within this scope. We are the first survey to provide a modular summary of methods and provide detailed analyses for each key module (*e.g.,* database content retrieval).
- We highlight the importance of *evaluating* NL2SQL *methods in a multi-angle way*, analyze the key NL2SQL error patterns, and provide a two-level error taxonomy.
- We provide practitioners with a roadmap for optimizing LLMs to NL2SQL task and a decision flow for selecting the suitable NL2SQL modules for various usage scenarios.

Contributions. We make the following contributions.

- *NL2SQL with Language Models.* We comprehensively review existing NL2SQL techniques from a lifecycle perspective (Figure 1). We introduce the NL2SQL task definition, discuss challenges (Figure 2), provide a taxonomy of NL2SQL solutions based on language models (Figure 3), and summarize the key modules of language model-powered NL2SQL solutions (Figure 5 and Table I). Next, we elaborate on each module of language model-powered NL2SQL methods, including the pre-processing strategies (Section IV), NL2SQL translation methods (Section V), and post-processing techniques (Section VI).
- *NL2SQL Benchmarks.* We review NL2SQL benchmarks based on their characteristics (Figure 10). We analyze each benchmark in-depth and present their statistical information (Table II). (Section VII)
- *NL2SQL Evaluation and Errors Analysis.* We highlight the importance of evaluation in developing practical NL2SQL solutions. We review widely used evaluation metrics and toolkits for assessing NL2SQL solutions. We provide a taxonomy to summarize typical errors produced by NL2SQL methods. (Section VIII)
- *Practical Guidance for Developing NL2SQL Solutions.* We provide a roadmap for optimizing existing LLMs to NL2SQL tasks. (Figure 13(a)). In addition, we design a decision flow to guide the selection of appropriate NL2SQL modules for different scenarios (Figure 13(b)).
- *Open Problems in NL2SQL.* Finally, we discuss new research opportunities, including the open-world NL2SQL problem and cost-effective NL2SQL solutions (Section X).
- *NL2SQL Handbook.* We maintain a continuously updated handbook[1] for readers to easily track the latest NL2SQL techniques in the literature and provide practical guidance for researchers and practitioners.

II. NL2SQL Problem and Background

In this section, we first formalize the definition of the NL2SQL task (Section II-A). We then introduce the workflow

[1] NL2SQL Handbook: https://github.com/HKUSTDial/NL2SQL_Handbook

of how humans perform the NL2SQL task (Section II-B) and discuss the key challenges (Section II-C). Finally, we describe the evolution of NL2SQL solutions based on the development of language models (Section II-D).

A. Problem Formulation

Definition 1 (Natural Language to SQL (NL2SQL)). Natural Language to SQL (NL2SQL), also known as Text-to-SQL, is the task of converting natural language queries (NL) into corresponding SQL queries (SQL) that can be executed on a relational database (DB). Specifically, given an NL and a DB, the goal of NL2SQL is to generate an SQL that accurately reflects the user's intent and returns the appropriate results when executed on the database.

Discussion. In some cases, the corresponding SQL to an NL query may be multiple due to the ambiguity or underspecification of the NL, or the ambiguity of the database schema. In addition, even when the NL, database schema, and database content are clear and specific, there may still be multiple equivalent SQLs that can satisfy the given NL query.

B. NL2SQL Human Workflow

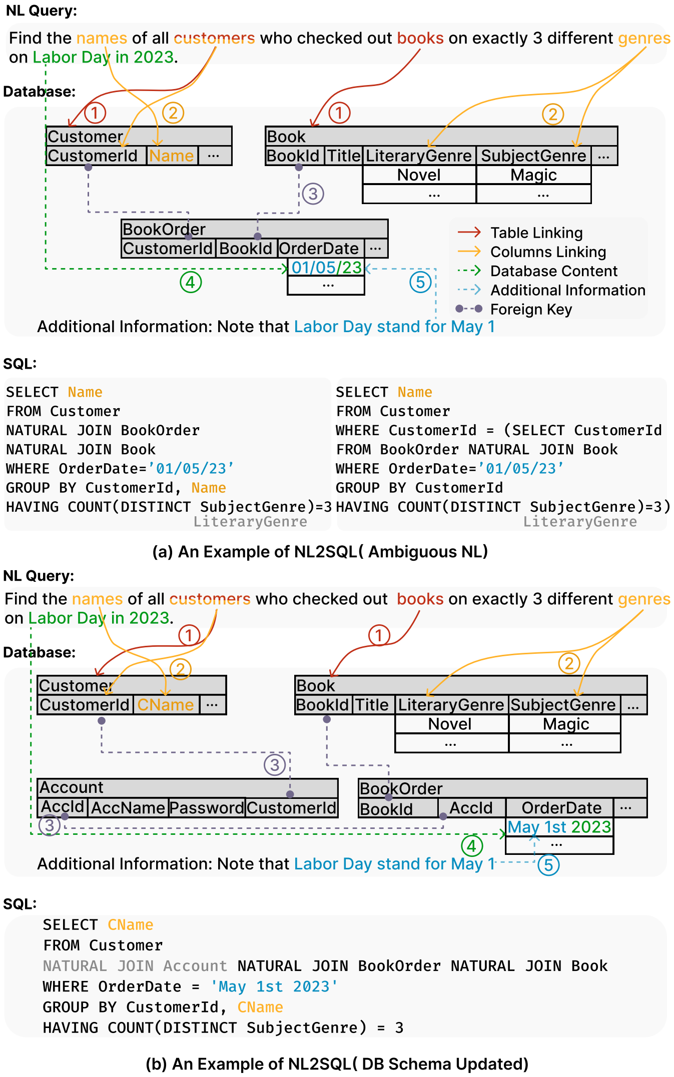
When humans, such as Database Administrator (DBA), perform the NL2SQL task, they first attempt to understand NL, then examine the database schema and contents, and finally write the corresponding SQL based on their SQL knowledge. Next, we will provide a detailed description of this process, as shown in Figure 2(a).

Step-1: Natural Language Query Understanding: Given the NL query "*Find the names of all customers who checked out books on exactly 3 different genres on Labor Day in 2023*", the first task is to comprehend the user's intent and identify the key components of the NL. To this end, the DBA may first identify some key terms and phrases in the given NL. For example, in this query, the key terms are: 1) *Entities or Attributes*: "names", "customers", "books", and "genres"; 2) *Temporal Context*: "Labor Day in 2023"; and 3) *Specific Conditions*: "exactly 3 different genres".

Then, the DBA may further understand the overall purpose of the NL query. In this case, the query intends to retrieve a list of customer names based on specific borrowing behavior on a particular date.

Step-2: Schema Linking and DB Content Retrieval: Next, the DBA examines the database schema and values to identify the relevant tables, columns, and cell values needed to produce the SQL. For example, the DBA may determine that the "Customer" and "Book" tables are relevant based on their understanding of the NL (see Figure 2(a)-①). The DBA then decides which columns should be mentioned. For example, the keyword "genres" can refer to either "LiteraryGenre" or "SubjectGenre" (see Figure 2(a)-②).

Furthermore, the DBA should interpret "Labor Day in 2023" based on the context. In the US, "Labor Day in 2023" refers to "September 4th, 2023", while in China, it refers to "May 1st, 2023". This judgment involves domain knowledge or available additional information (see Figure 2(a)-⑤).



NL Query:

Find the names of all customers who checked out books on exactly 3 different genres on Labor Day in 2023.

Additional Information: Note that Labor Day stand for May 1

SQL:

```
SELECT Name
FROM Customer
NATURAL JOIN BookOrder
NATURAL JOIN Book
WHERE OrderDate='01/05/23'
GROUP BY CustomerId, Name
HAVING COUNT(DISTINCT SubjectGenre)=3
                      LiteraryGenre
```

```
SELECT Name
FROM Customer
WHERE CustomerId = (SELECT CustomerId
FROM BookOrder NATURAL JOIN Book
WHERE OrderDate='01/05/23'
GROUP BY CustomerId
HAVING COUNT(DISTINCT SubjectGenre)=3)
                      LiteraryGenre
```

(a) An Example of NL2SQL(Ambiguous NL)

NL Query:

Find the names of all customers who checked out books on exactly 3 different genres on Labor Day in 2023.

Additional Information: Note that Labor Day stand for May 1

SQL:

```
SELECT CName
FROM Customer
NATURAL JOIN Account NATURAL JOIN BookOrder NATURAL JOIN Book
WHERE OrderDate = 'May 1st 2023'
GROUP BY CustomerId, CName
HAVING COUNT(DISTINCT SubjectGenre) = 3
```

(b) An Example of NL2SQL(DB Schema Updated)

Fig. 2: Examples of the NL2SQL Task and Its Challenges.

Step-3: Translating the NL Intent into the SQL: Finally, the DBA writes the corresponding SQL based on their understanding of the NL and the results of schema linking and database content retrieval. This process, known as "NL2SQL Translation", requires the DBA to leverage their SQL knowledge and understanding of database concepts. However, this process can be very challenging due to the ambiguity of the NL or the complexity of the database. For example, as illustrated in Figure 2(a), despite understanding the need to link the *Customer* and *Book* tables, one must be familiar with the usage and norms of employing either a natural join or a subquery. Additionally, there may be multiple corresponding SQL queries because "genres" can refer to either "LiteraryGenre" or "SubjectGenre".

C. NL2SQL Task Challenges

As mentioned in Section II-B, there are three key steps in the NL2SQL task. From these steps, we can identify three *inherent* challenges: the uncertainty of the natural language, the complexity of the database, and the translation from the "free-form" natural language queries to the "constrained and formal" SQL queries.

In this section, we will discuss these fundamental challenges of the NL2SQL task. We will then analyze the *technical* challenges, *i.e.,* the challenges we face when developing a strong NL2SQL solution in real-world scenarios.

C1: The Uncertain Natural Language Query. Natural language often contains uncertainties due to ambiguity and underspecification [30]. In NL2SQL tasks, the challenges related to NL can be summarized as follows:

- *Lexical Ambiguity*: This occurs when a single word has multiple meanings. For example, the word "bat" can refer to an animal or a *baseball bat* (noun) or the action of *swinging* (verb).
- *Syntactic Ambiguity*: This occurs when a sentence can be parsed in multiple ways. For example, in the sentence "Mary saw the man with the telescope", the phrase "with the telescope" can mean either that Mary used a telescope to see the man or that the man had a telescope.
- *Underspecification*: This occurs when linguistic expressions lack sufficient detail to convey specific intentions or meanings clearly. For example, "Labor Day in 2023" refers to September 4th in the US but May 1st in China.
- *User Mistakes*: Spelling mistakes and grammatical errors significantly increase the complexity of natural language understanding.

C2: The Complex Database and Dirty Content. The NL2SQL task requires an in-depth understanding of the database schema, including table names, column names, and the interrelations between tables, as well as the database context involving the attributes and values of data. The complexity of modern database schema and the vast volume of data present substantial challenges to the effective operation of NL2SQL tasks. These complexities include:

- *Complex Relationships Among Numerous Tables*: A database can include hundreds of tables interconnected through complex relationships. NL2SQL systems must be able to understand and utilize these relationships accurately when generating SQL queries.
- *Similarity in Column Names*: Managing columns with similar names across different tables is a common challenge in the NL2SQL task. For example, a database might contain multiple tables each with a column named "date", where one might represent the "creation date" while another might represent the "expiration date".
- *Domain-Specific Schema Variations*: Different fields may have unique database designs, which means an NL2SQL system must adapt to each specific schema's difference, making generic solutions challenging. For example, table and column names are often expressed using abbreviations or vague expressions in the finance domain.
- *Large and Dirty Values*: In the context of large databases, handling the immense volume of data efficiently is essential, as it is impractical to use all data as input. The system must focus on effectively extracting and formatting relevant data from queries to meet database requirements. In addition, it's crucial for the system to have fault tolerance capabilities to manage and mitigate errors or inconsistencies present in real-world databases, ensuring the reliability and accuracy of the query outputs.

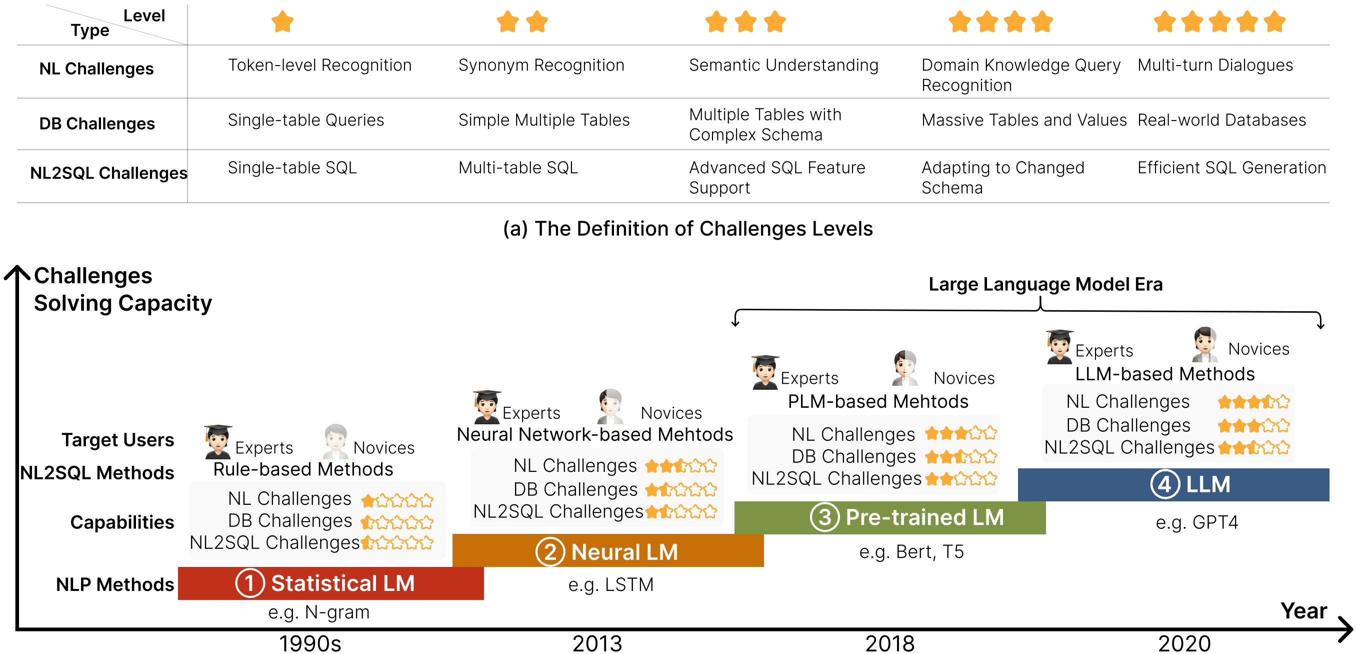
C3: NL2SQL Translation. The NL2SQL task differs from the compilation of a high-level programming language to a low-level machine language, as it usually has a *one-to-many* mapping between the input NL and output SQL queries. Specifically, the NL2SQL task faces several unique challenges:

- *Free-form NL vs. Constrained and Formal SQL*: the SQL queries follow a strict syntax, whereas natural language is more flexible and varied. Translating NL queries into corresponding SQL queries requires adherence to SQL syntax rules to ensure they are executable.
- *Multiple Possible SQL Queries*: a single NL query can correspond to multiple SQL queries that satisfy the requirements, leading to ambiguity in determining appropriate SQL translation (see the example in Figure 2(a)).
- *Database Schema Dependency*: the NL2SQL translation process is significantly dependent on the database schema it interacts with. As shown in Figure 2 (a) and (b), for the same NL query (intent), a change in the database schema will correspond to different SQL queries. This requires NL2SQL solutions to dynamically adapt their translations to different databases.

Beyond the above intrinsic challenges of translating natural language queries to SQL, developers must also navigate several technical obstacles to build reliable and efficient NL2SQL systems. Next, we will discuss several technical challenges that need to be addressed to develop strong NL2SQL solutions.

Technical Challenges in Developing NL2SQL Solutions. Developing robust NL2SQL solutions involves addressing several technical challenges. These include:

- *Efficiency of the Model*: Ensuring the model can process natural language queries and convert them to SQL efficiently, minimizing latency. This consideration is crucial as efficiency directly impacts user experience and operational costs, especially in scenarios requiring low latency.
- *Efficiency of SQL*: The SQL generated by NL2SQL models need to be not only correct but also optimized for performance. This involves optimization in the selection of joins, indexes, and query structures. Efficient queries reduce the load on databases, enhancing the system's responsiveness and throughput.
- *Cost-effective Solution*: Deploying NL2SQL models, especially those using large language models, requires significant resources, including hardware and API costs. These models are expensive to run and can consume a lot of cost (such as energy and monetary cost). Efficient resource management is crucial to keep operational costs down and reduce environmental impact.
- *Insufficient and Noisy Training Data*: Acquiring high-quality NL2SQL training data is extremely challenging. The limited amount of publicly available training data is often insufficient for training robust models. Additionally, the quality of this data is frequently compromised by noisy annotations. Annotators need database knowledge,



Type \ Level	★	★★	★★★	★★★★	★★★★★
NL Challenges	Token-level Recognition	Synonym Recognition	Semantic Understanding	Domain Knowledge Query Recognition	Multi-turn Dialogues
DB Challenges	Single-table Queries	Simple Multiple Tables	Multiple Tables with Complex Schema	Massive Tables and Values	Real-world Databases
NL2SQL Challenges	Single-table SQL	Multi-table SQL	Advanced SQL Feature Support	Adapting to Changed Schema	Efficient SQL Generation

(a) The Definition of Challenges Levels

(b) The Evolution of NL2SQL Solutions

Fig. 3: The Evolution of NL2SQL Solutions from the Perspective of Language Models.

which increases costs, and the complexity of the NL2SQL task often leads to errors, for example, in the Spider dataset [31], which required around 1100 hours of human labor to annotate about 11K (NL, SQL) pairs. However, it was subsequently identified that about 4% of the pairs had annotation errors, requiring additional annotation efforts to correct them.

- *Data Privacy*: Data privacy is crucial for NL2SQL systems, especially when the databases contain sensitive data. Using cloud-based APIs like GPT-4 increases privacy risks, as it involves sending data to external servers.
- *Trustworthiness and Reliability*: For NL2SQL models to be widely used, they need to be trustworthy and reliable. This means they must consistently produce accurate results across different datasets and use cases. Trustworthiness also requires that the model's decisions are transparent. This allows users to understand and check the SQL it generates. For example, using explainable AI to show how decisions are made and developing strong evaluation metrics to assess accuracy.

D. Challenges Solving with Large Language Models

The evolution of NL2SQL technology has been marked by substantial advancements over the years, driven primarily by progress in Natural Language Processing (NLP).

In Figure 3(a), we categorize the challenges of NL2SQL into five levels and define each level's specific challenges. The first three levels focus on challenges that have been addressed or are still being tackled, affirming the progressive development of NL2SQL. The fourth level symbolizes the challenges we aim to resolve in the LLMs stage. Finally, the fifth level represents our aspirations for the ultimate NL2SQL system.

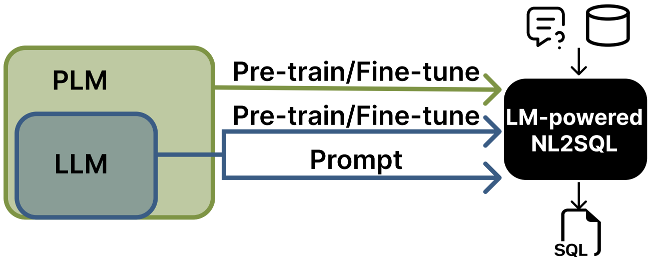


Fig. 4: The Categorization of PLM and LLM in NL2SQL.

In Figure 3(b), we describe the evolution of NL2SQL solutions from the perspective of language models, categorizing it into four stages: the rule-based stage, the neural network-based stage, the PLM-based stage, and the LLM-based stage. For each stage of NL2SQL, we analyze the changes in target users and the extent to which challenges are addressed.

Remark: PLM vs. LLM. Figure 4 illustrates the differences between LLMs and PLMs. The LLMs are a type of PLMs characterized by superior language understanding and emergent capabilities [32], [33]. These emergent capabilities enable it to perform NL2SQL tasks using prompts. However, employing PLMs for NL2SQL tasks typically requires pre-training or fine-tuning to achieve similar performance.

1) Rule-based Stage: In the early stages of NL2SQL technology development, research primarily focused on rule-based methods. These methods [30], [34]–[36] utilized predefined rules or semantic parsers to understand natural language queries and convert them into SQL queries. For example, NaLIR [35] utilizes a syntactic parser to understand the NL query and link it to database elements. Next, it relies on manually crafted rules to generate the corresponding SQL query. However, rule-based methods have limitations in terms of adaptability, scalability, and generalization capabilities. At this stage, natural language understanding is limited to the token level, and research focuses on single-table databases. (see Figure 3(b)-①)

2) Neural Network-based Stage: To alleviate these limitations, researchers turned to leveraging neural networks to solve the NL2SQL task. Methods based on sequence-to-sequence models or graph neural networks (GNNs) are developed [37]–[39]. For example, given the various expressions possible in the `WHERE` clause of SQL queries, Seq2SQL [40] employs a reinforcement learning approach for partial training of this clause. This method trains the aggregation functions and `SELECT` clause using cross-entropy loss while employing a reward function to train the `WHERE` clause. Addressing the potential challenges of reinforcement learning, SQLNET [41] proposes a query skeleton technique using fixed slots, transforming the problem into a classification task. However, the generalization ability of these methods is limited by the model size and the amount of training data. At this stage, natural language understanding has advanced with the development of neural networks, not only handling synonyms and synonymous phrases but even beginning to comprehend simple text. Consequently, the research focus shifted from single-table databases like WikiSQL [40] to more complex databases like Spider [31] that involve multiple tables. (see Figure 3(b)-②)

3) PLM-based Stage: With the introduction of PLMs such as BERT [42] and T5 [43] in 2018, NL2SQL methods based on PLMs [7], [44], [45] have achieved competitive performance on various benchmark datasets. These models brought significant advancements in natural language understanding and generation, enabling more accurate and efficient NL2SQL translation. In this stage, fine-tuning the PLMs is often required to adapt to NL2SQL tasks. For example, Graphix-T5 [44] is a hybrid model that combines a conventional pre-trained transformer architecture with custom graph-aware layers designed to enhance its performance on tasks involving graph-structured data. However, these models still face challenges in handling complex database schema and often require specific training for NL2SQL tasks. At this stage, PLMs trained on extensive corpora have further enhanced natural language understanding capabilities, successfully resolving approximately 80% of the cases in the Spider database. However, when addressing external hard-level cases, the accuracy is only around 50% [46] (see Figure 3(b)-③).

4) LLM-based Stage: LLMs demonstrate unique emergent capabilities, which enable LLMs to surpass traditional PLMs in NLP tasks. This development has also triggered a new wave of solutions for the NL2SQL task. These LLM-based NL2SQL methods have become the most representative solutions in the current NL2SQL landscape [5], [6], [47], [48]. Current optimizations primarily focus on prompt design [6] and training LLMs [47]. DAIL-SQL [6] utilizes the GPT-4 model through effective prompt engineering techniques and has achieved competitive results on the Spider dataset. CodeS [47] is making attempts to create language models specifically for the NL2SQL task. By incrementally pretraining on StarCoder [49] with a large corpus related to NL2SQL tasks, CodeS has shown exceptional performance across numerous challenging NL2SQL benchmarks. At this stage, the emergence of emergent capabilities has significantly enhanced natural language understanding. Consequently, the focus of challenges has shifted more towards the database layer. The introduction of benchmarks like Bird [50] and Bull [48] reflects a heightened interest in addressing NL2SQL solutions under conditions of massive tables and values, as well as designing NL2SQL solutions for specific domains (see Figure 3(b)-④).

Although the LLM-based NL2SQL methods are still in their early phases, they have already demonstrated impressive performance. This not only validates the effectiveness of these models but also highlights the immense potential of this approach. With continuous advancements and optimization of the LLM-based methods, we believe these methods will be able to address more complex issues in practical applications.

NL2SQL Solution in the Era of LLMs. Broadly speaking, there are two major approaches to leverage the capabilities of LLMs for NL2SQL: 1) in-context learning, and 2) pre-train/fine-tune LLMs specialized for NL2SQL.

In-Context Learning for NL2SQL. For in-context learning NL2SQL methods, the goal is to optimize the prompt function P to guide the LLMs, which can be formulated as follows:

$$\mathcal{F}_{\text{LLM}}(P \mid \textsc{nl}, \textsc{db}, \textsc{k}) \rightarrow \textsc{sql},$$

where K denotes additional information or domain-specific knowledge related to NL or DB. P is a *prompt function* that transforms the input (NL, DB, K) into a suitable *textual prompt* for the LLMs. An appropriately designed P can effectively guide the LLMs to perform the NL2SQL task more accurately.

Employing in-context learning strategies for NL2SQL does not involve optimizing the parameters of the LLMs, thereby treating the LLMs as *off-the-shelf* tools. If the user has sufficient training data or hardware resources to calibrate the parameters of the LLMs, model performance and accuracy can be enhanced by tailoring it more closely to the specific NL2SQL task.

Pre-train and Fine-tune LLMs for NL2SQL. Fully optimizing the parameters of LLMs for NL2SQL tasks involves two critical stages: pre-train and fine-tune, which can be formulated as follows:

$$\text{LLM}^* = \mathcal{F}_{\text{fine-tune}}\left(\mathcal{F}_{\text{pre-train}}(\text{LLM}, \mathcal{D}_p), \mathcal{D}_f\right)$$

During pre-train, the LLM is trained on a large-scale and diverse dataset $\mathcal{D}_p$ that includes a broad range of linguistic patterns and domain-general knowledge, enabling the model to develop robust understanding capabilities.

In the subsequent fine-tuning stage, the pre-trained model is further adjusted on a more specialized dataset $\mathcal{D}_f$, which is closely aligned with the NL2SQL task. This targeted training refines the model's capabilities, enabling it to more effectively interpret and generate SQL based on natural language queries.

III. Language Model-powered NL2SQL Overview

We summarize the key modules of NL2SQL solutions utilizing language models, especially PLMs and LLMs, as illustrated in Figure 5. Additionally, we compare the key module differences of existing NL2SQL solutions in Table I.

A. Pre-Processing

Pre-processing serves as an enhancement to the model's inputs in the NL2SQL parsing process. Although not strictly necessary, pre-processing significantly contributes to the refinement of NL2SQL parsing [51].

- Schema Linking: This key module identifies the most relevant tables and columns from NL2SQL (Section IV-A).
- Database Content Retrieval: This key module accesses the appropriate database contents or cell values needed for formulating SQL (Section IV-B).
- Additional Information Acquisition: This key module enriches the contextual backdrop by integrating domain-specific knowledge (Section IV-C).

B. NL2SQL Translation Methods

NL2SQL translation methods constitute the core of the NL2SQL solution, responsible for converting input natural language queries into SQL queries. Existing methods relevant to NL2SQL translation can be summarized as follows:

- Encoding Strategy: This key module converts the input NL and database schema into an internal representation, capturing the semantic and structural information of the input data (Section V-A).
- Decoding Strategy: This key module transforms the internal representation into SQL queries (Section V-B).
- Task-specific Prompt Strategy: This module provides tailored guidance for the NL2SQL model, optimizing the NL2SQL translation workflow (Section V-C).
- Intermediate Representation for NL2SQL Translation: This module serves as a bridge between NL and SQL translation, providing a structured approach to abstract, align, and optimize NL understanding, simplify complex reasoning, and guide the generation of accurate SQL queries (Section V-D).

C. Post-Processing

Post-processing is a crucial step to refine the generated SQL queries, ensuring they meet user expectations more accurately. This involves enhancing the initial SQL output using various strategies, as discussed below.

- SQL Correction Strategies: This module aims to identify and correct syntax errors in generated SQL queries. (Section VI-A).
- Output Consistency: This module ensures the uniformity of SQL queries by sampling multiple reasoning results and selecting the most consistent result. (Section VI-B).
- Execution-Guided Strategies: This module uses the execution results of SQL queries to guide subsequent refinements. (Section VI-C).
- N-best Rankers Strategies: N-best re-ranking strategies reorder the top results from the model to improve query accuracy. (Section VI-D).

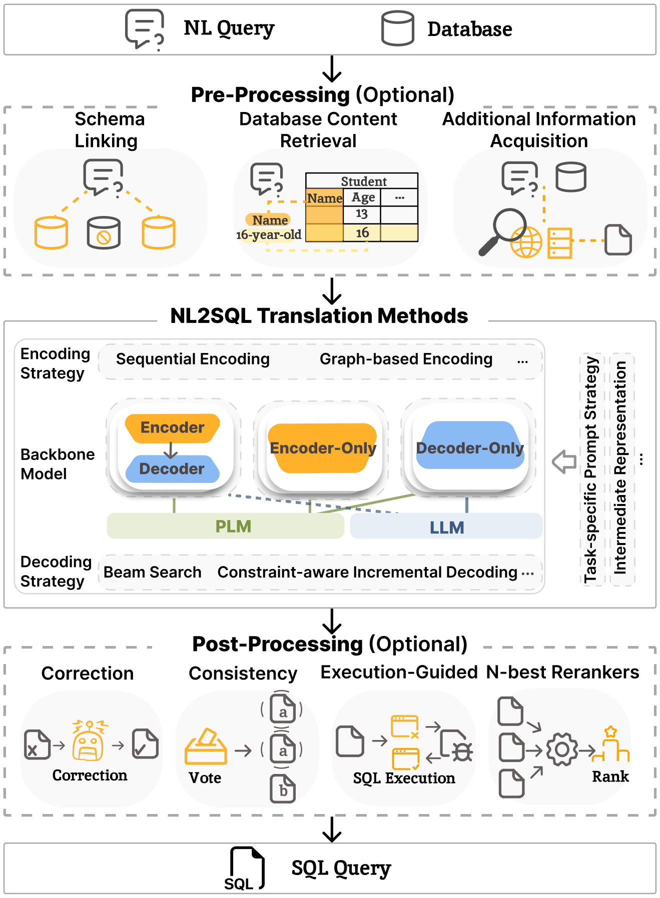


Fig. 5: An Overview of NL2SQL Modules in the LLM Era.

IV. Pre-Processing Strategies for NL2SQL

The pre-processing step is crucial for the NL2SQL translation process because it identifies the most relevant tables and columns (*i.e.,* Schema Linking) and retrieves suitable database contents/cell values (*i.e.,* DB Content Retrieval) used for generating SQL queries in advance. What's more, it enriches context by adding domain-specific knowledge (*i.e.,* Additional Information Acquisition), enhances efficiency by focusing on relevant data, improves the understanding of the query context, and corrects errors to prevent their propagation.

A. Schema Linking

The purpose of the schema linking is to identify the tables and columns related to the given NL query. It ensures the accurate mapping and processing of key information within the limited input, thereby improving the performance of the NL2SQL task. In the LLMs era, schema linking has become increasingly crucial due to the input length limit of LLMs.

We categorize existing schema-linking strategies into three groups based on their characteristics: *1) string matching-based methods*, *2) neural network-based methods*, and *3) in-context learning-based for schema linking*.

1) String Matching-based Schema Linking: Early research [38], [78], [79] primarily concentrated on string matching. String matching-based schema linking utilizes similarity

TABLE I: Comparisons of Existing NL2SQL Solutions.

Methods	Years	Finetuning	Pre-Processing			NL2SQL Translation Methods					Post-Processing			
			Schema Linking	DB Content Retrieval	Additional Information Acquisition	Backbone Model	Encoding Strategy	Intermediate Representation	Task-specific Prompt Strategy	Decoding Strategy	Correction	Consistency	Execution -Guided	N-best Rerankers
CHESS [52]	2024	-	✓	✓	✓	Decoder-Only	Sequential Encoding	-	COT	Greedy Search	✓	✓	✓	-
CodeS [47]	2024	-	✓	✓	-	Decoder-Only	Sequential Encoding	-	-	Greedy Search	-	-	✓	-
SFT CodeS [47]	2024	✓	✓	✓	✓	Decoder-Only	Sequential Encoding	-	-	Greedy Search	-	-	✓	-
FinSQL [48]	2024	✓	✓	-	✓	Decoder-Only	Sequential Encoding	-	-	Greedy Search	✓	✓	-	-
DTS-SQL [53]	2024	✓	✓	-	-	Decoder-Only	Sequential Encoding	-	-	Greedy Search	-	-	-	-
TA-SQL [54]	2024	-	✓	-	-	Decoder-Only	Sequential Encoding	Sketch Structure	COT	Greedy Search	-	-	-	-
SuperSQL [46]	2024	-	✓	✓	✓	Decoder-Only	Sequential Encoding	-	-	Greedy Search	-	✓	-	-
ZeroNL2SQL [45]	2024	✓	-	-	-	Encoder-Decoder	Sequential Encoding	Sketch Structure	Decomposition	Beam Search	✓	-	✓	-
PET-SQL [55]	2024	✓	✓	-	✓	Decoder-Only	Sequential Encoding	-	-	Greedy Search	-	✓	-	-
CoE-SQL [56]	2024	-	-	-	✓	Decoder-Only	Sequential Encoding	-	CoT	Greedy Search	✓	-	-	-
PURPLE [57]	2024	-	✓	-	✓	Decoder-Only	Sequential Encoding	-	-	Greedy Search	✓	✓	✓	-
MetaSQL [58]	2024	-	✓	-	✓	Decoder-Only	Sequential Encoding	-	Decomposition	Greedy Search	-	-	-	✓
DEA-SQL [59]	2024	-	✓	-	✓	Decoder-Only	Sequential Encoding	-	Decomposition	Greedy Search	✓	-	-	-
DIN-SQL [5]	2023	-	✓	-	✓	Decoder-Only	Sequential Encoding	Syntax Language	Decomposition	Greedy Search	✓	-	-	-
DAIL-SQL [6]	2023	-	-	-	✓	Decoder-Only	Sequential Encoding	-	-	Greedy Search	-	✓	-	-
C3-SQL [60]	2023	-	✓	-	-	Decoder-Only	Sequential Encoding	-	COT	Greedy Search	-	✓	-	-
RESDSQL [7]	2023	✓	✓	✓	-	Encoder-Decoder	Sequential Encoding	Syntax Language	Decomposition	Beam Search	-	-	-	-
T5-3B+NatSQL+Token Preprocessing [61]	2023	✓	✓	✓	-	Encoder-Decoder	Sequential Encoding	Syntax Language	-	Greedy Search	-	-	-	-
ACT-SQL [62]	2023	-	✓	-	✓	Decoder-Only	Sequential Encoding	-	CoT	Greedy Search	-	-	-	-
ODIS [63]	2023	-	-	-	✓	Decoder-Only	Sequential Encoding	-	-	Greedy Search	-	-	-	-
MAC-SQL [64]	2023	-	✓	-	-	Decoder-Only	Sequential Encoding	-	Decomposition	Greedy Search	✓	-	✓	-
SC-Prompt [1]	2023	✓	-	-	-	Encoder-Decoder	Separate Encoding	Sketch Structure	-	Beam Search	✓	-	-	-
CatSQL [65]	2023	✓	-	-	-	Encoder-Only	Sequential Encoding	Sketch Structure	-	Beam Search	✓	-	-	-
SQLFormer [66]	2023	✓	✓	-	-	Encoder-Decoder	Graph-based Encoding	-	-	Beam Search	-	-	-	-
G^{3}R [67]	2023	✓	✓	✓	-	Encoder-Only	Graph-based Encoding	-	COT	Beam Search	-	-	-	✓
Graphix-T5 [44]	2022	✓	✓	✓	-	Encoder-Decoder	Graph-based Encoding	-	-	Constraint-aware Incremental	-	-	-	-
SHiP [68]	2022	✓	-	✓	-	Encoder-Decoder	Graph-based Encoding	-	-	Constraint-aware Incremental	-	-	-	-
N-best List Rerankers [69]	2022	✓	✓	✓	-	Encoder-Decoder	Sequential Encoding	-	-	Constraint-aware Incremental	-	-	-	✓
RASAT [70]	2022	✓	-	✓	-	Encoder-Decoder	Graph-based Encoding	-	-	Constraint-aware Incremental	-	-	-	-
PICARD [71]	2022	✓	-	✓	-	Encoder-Decoder	Sequential Encoding	-	-	Constraint-aware Incremental	-	-	-	-
TKK [72]	2022	✓	-	✓	-	Encoder-Decoder	Separate Encoding	Sketch Structure	Decomposition	Constraint-aware Incremental	-	-	-	-
S^{2}SQL [73]	2022	✓	✓	✓	-	Encoder-Only	Graph-based Encoding	-	-	Greedy Search	-	-	-	-
RAT-SQL [74]	2021	✓	✓	✓	-	Encoder-Only	Graph-based Encoding	Syntax Language	-	Beam Search	-	-	-	-
SmBoP [75]	2021	✓	-	✓	-	Encoder-Only	Graph-based Encoding	-	-	Beam Search	-	-	-	-
RaSaP [76]	2021	✓	✓	✓	-	Encoder-Only	Graph-based Encoding	-	-	Beam Search	-	-	-	-
BRIDGE [77]	2020	✓	-	✓	-	Encoder-Only	Sequential Encoding	-	-	Others	-	-	-	-

measures between the text in the NL and the schema elements (*e.g.,* table and column names) to identify relevant mappings. These methods rely on exact or approximate matching techniques to align the NL with the database schema, ensuring that the query components are correctly associated with the corresponding database elements. Exact matching, as employed by IRNet [78], is the simplest approach for identifying matches. An exact match requires candidates to be identical, while a partial match occurs when one candidate is a substring of the other. This method can uncover obvious links but may result in false positives when candidates share common words. Approximate string matching, such as the Damerau–Levenshtein distance [80] used by ValueNet [81], is another useful technique. It helps identify matches with different spellings or spelling mistakes. However, this methods lack the ability to handle synonyms and is not robust to variations in vocabulary.

2) Neural Network-based Schema Linking: To alleviate the limitations of traditional string matching-based methods, some researchers employ deep neural networks to match database schema with natural language queries [7], [51], [74], [82]. These approaches effectively parse complex semantic relationships between language and database structures.

DAE [82] formulates schema linking as a sequential tagging problem and proposes a two-stage anonymization model to learn the semantic relationship between schema and NL. However, DAE does not demonstrate how schema linking impacts the performance of NL2SQL tasks, as the lack of an annotated corpus. To address this, SLSQL [51] annotates the schema linking information for each instance in the training and development sets of Spider [31] to support a data-driven and systematic study. With the advent of the transformer, researchers start to explore the use of attention mechanisms and PLMs for schema linking. RAT-SQL [74] employs a relation-aware self-attention mechanism to integrate global reasoning across schema entities and question terms with structured reasoning based on predefined schema relationships. RESDSQL [7] proposes a ranking-enhanced encoding framework for schema linking. An additional cross-encoder is trained to classify tables and columns based on the input query. This framework ranks and filters them according to classification probabilities, resulting in a ranked sequence of schema items.

However, neural network-based schema linking methods often struggle to generalize effectively across databases with significantly different schema or domains, especially when training data are limited.

3) In-Context Learning for Schema Linking: With the advancement of LLMs such as GPT-4, existing research aims to harness the strong reasoning capabilities of LLMs for schema linking, *i.e.,* directly identifying and linking relevant database schema components from the NL query. One important technique is to utilize the In-Context Learning (ICL) technique [83].

This approach leverages the capability of LLMs to understand and process complex language patterns and relationships within the data schema, facilitating a more dynamic and flexible schema linking process [5], [52], [60], [64], [84].

DIN-SQL [5] designs a prompt-based module for schema linking. The prompt comprises ten randomly chosen samples from the training set of the Spider and follows the Chain-of-Thought [85] strategy. For each column mentioned in the NL, the corresponding columns and their tables are selected from the database schema. C3 [60] designs different zero-shot prompts to instruct GPT-3.5 for table and column linking, employing the self-consistency method. For the table linking, the prompt guides the process in three steps: ranking tables by relevance, ensuring all relevant tables are included, and outputting in list format. For the column linking, another prompt guides the ranking of columns within candidate tables and outputting in dictionary format, prioritizing those matching question terms or foreign keys. Similarly, MCS-SQL [84] also performs schema linking in two steps involving table linking and column linking. However, MCS-SQL distinguishes itself by utilizing multiple prompts in both steps to maximize recall. MAC-SQL [64] propose a multi-agent collaborative framework for NL2SQL. The *Selector* agent performs the schema linking task. The *Selector* is activated only when the length of the database schema prompt exceeds a specified threshold. CHESS [52] utilizes GPT-4 to extract keywords from both NL and Evidence (additional information provided by BIRD [50]). By designing different prompts, it implements an efficient three-stage schema pruning protocol.

Employing ICL for schema linking has demonstrated commendable performance. However, it is important to note that LLMs face inherent limitations in the length of context they can process. Complex schema with many tables and columns may exceed this limitation. In addition, the effectiveness of ICL is heavily dependent on the quality of the prompts.

B. Database Content Retrieval

The purpose of database content retrieval is to efficiently retrieve cell values through textual searching algorithms and database indexing. Unlike schema linking, which focuses on finding relevant tables and columns based on the NL query, database content retrieval emphasizes efficiently retrieving cell values. Given the large scale of databases, retrieving cell values from them is resource-intensive and poses potential risks of exposing sensitive data [77]. Therefore, it is crucial to implement appropriate strategies for the scenario requirement.

We categorize existing database content retrieval strategies into three groups based on their characteristics: *1) String Matching-based Methods*, *2) Neural Network-based Methods*, and *3) Index Strategy for Database Content Retrieval*.

1) String Matching-based Methods: String matching-based methods are used to find and compare sequences of cell values related to the given NL query through string matching.

IRNet [78] uses the n-grams method and regards those begin and end with quotes as cell values. Besides n-grams, ValueNet [81] implements two other methods for generating candidate cell values based on string similarity and heuristic selection. RESDSQL [7], Graphix-T5 [44], and PICARD [71] use the Longest Common Substring algorithm [86], which determines the maximum length sequence of characters that appear in each given string.

Further advancing this approach, BRIDGE [77] designs an anchor text matching to extract cell values mentioned in the NL automatically. It uses a heuristic method to calculate the maximum sequence match between the problem and the cell values to determine the matching boundary. When the cell values are substrings of words in the query, the heuristic can exclude those string matches. The matching threshold is then adjusted by making coarse accuracy measurements.

Building on the concepts introduced by BRIDGE, various frameworks such as RESDSQL [7], RASAT [70], SuperSQL [46], and SmBoP [75] follow the same database content module as BRIDGE [77]. While string matching-based methods are direct and effective, they face challenges such as dealing with synonyms. In addition, when there is a large amount of database content, the high computational cost cannot be ignored.

2) Neural Network-based Methods: Neural networks can learn complex data formats and semantic representations through multiple layers of nonlinear transformations to capture semantic features, thus alleviating the synonym issues.

TABERT [87] utilizes a method called *database content snapshots* to encode the relevant subset of database content corresponding to the NL query. It uses an attention mechanism to manage information between cell value representations across different rows. Another common approach is leveraging the relationships in graphs to represent database content. For example, IRNet [78] employs the knowledge graph ConceptNet [88] to recognize cell value links and search cell value candidates in the knowledge graph. When a result exactly or partially matches a cell value, the column is assigned a type of value exact match or partial match, respectively.

RAT-SQL [74] further improves structural reasoning capabilities by modeling the relationship between cell values and the NL query. Specifically, it identifies the column-value relationship, meaning that the value in the question is part of the candidate cell value of the column. Graphix-T5 [44] and LGESQL [89] use a value-match relation, which defines the relationship between cell values and questions similarly to RAT-SQL.

Despite their ability to capture semantic features, these methods might still struggle with ambiguous or context-dependent NL, potentially leading to inaccurate cell value retrieval. In addition, the complexity of neural network architectures would demand substantial computational resources.

3) Index Strategy for Database Content Retrieval: Efficiently retrieving relevant cell values from a database is crucial for the performance of NL2SQL systems, especially when dealing with large datasets. Therefore, existing research has employed indexing as a crucial method for improving the efficiency of database content retrieval, as it allows faster access to relevant cell values [47], [52].

CHESS [52] utilizes a Locality-sensitive Hashing algorithm [90] for approximate nearest neighbor searches. It indexes unique cell values to quickly identify the top similar values related to the NL query. This approach significantly speeds up the process of computing the edit distance and semantic embedding between the NL query and cell values.

CodeS [47] introduces a coarse-to-fine cell value matching approach. It leverages indexes for a coarse-grained initial search, followed by a fine-grained matching process. First, it builds the index for all values using BM25 [91]. The index identifies candidate values relevant to NL. The Longest Common Substring algorithm [86] is then used to calculate the matching degree between NL and the candidate values to find the most relevant cell values.

While indexing strategies can significantly enhance the efficiency of searching for relevant cell values, they also have limitations. Index building requires an investment of time, and if the cell values in the database change frequently, the index must be continuously updated, introducing additional inference overhead.

C. Additional Information Acquisition

Additional information (*e.g.,* domain knowledge) plays an essential role in improving the comprehension capabilities of NL2SQL models for understanding the NL query, performing the schema linking, and benefiting the NL2SQL translation. This information can provide demonstration examples, domain knowledge, formulaic evidence, and format information for the NL2SQL backbone model or specific modules, thereby enhancing the quality of the generated results.

With the advancement of LLMs and in-context learning techniques, researchers often incorporate additional information as part of the textual inputs (prompts) along with few-shot examples. For example, DIN-SQL [5] inserts additional information through few-shot learning across multiple stages of the workflow, such as schema linking, query classification, task decomposition, and self-correction. These stages allow DIN-SQL to effectively tackle various challenges, including the complexity of schema links, identification of multiple table joins, and handling of nested queries. Similarly, CodeS [47] utilizes metadata examples of cross-domain databases as the main additional information, including data types and annotation text, which help the model resolve potential ambiguity issues and understand entity relationships. This extracted information is transformed into coherent text and concatenated with the question query to form the final input context. Other works that use similar additional information acquisition strategies include ODIS [63] and CHESS [52].

However, due to the large scale of domain knowledge bases and human-built examples, retrieving matching knowledge and suitable few-shot samples from the candidate pool greatly increases the usage of tokens, impacting inference efficiency and computational cost [5], [6]. To overcome these challenges and improve the accuracy and efficiency of information retrieval, some researchers apply similarity-based information retrieval mechanisms for additional information acquisition. For example, PET-SQL [55] constructs a pool of examples from the training set, which contains question frames and question-SQL pairs. Then, it selects the k examples that are most similar to the target question. These selected examples are combined with customized prompts as the final input.

Further refining this approach, DAIL-SQL [6] intricately designs a two-stage representation algorithm for additional information. It begins by presenting the question and database as SQL statement hints, thereby providing comprehensive database information. Following this, it employs a masking mechanism and similarity calculation to select appropriate examples and systematically organizes tags to enhance the efficiency of the algorithm. Building on this, SuperSQL [46] extends the representation algorithm of the DAIL-SQL by integrating similarity-based sample selection with schema linking and database content information, which filters out irrelevant schema, thereby enhancing the quality of SQL generation.

In addition to these methods, databases from real-world scenarios usually comprise a large amount of cross-domain knowledge or evidence. For example, BIRD [50] contains knowledge obtained from 95 databases and 37 specialized domains derived from real-life scenarios. This additional information plays an important role in the related NL2SQL studies [5], [6], [46], [47], [52], [53], [64].

Some other databases do not directly provide the corresponding additional information text, which requires researchers to develop extraction mechanisms to retrieve this information and convert it into natural language text. For example, REGROUP [92] constructs a formulaic knowledge base encompassing various domains, such as finance, real estate, and transportation. It leverages a Dense Passage Retriever (DPR) [93] to compute similarity scores for the retrieval results from the formulaic knowledge base. Subsequently, an Erasing-Then-Awakening (ETA) model is used to integrate the entities in these formulaic knowledge items with the entities in NL and schema. This model filters irrelevant entities below a confidence threshold and maps the remainder to schema elements, thereby grounding knowledge for accurate SQL query generation. ReBoost [94] engages with the LLMs model using the Explain-Squeeze Schema Linking mechanism. This mechanism is a two-phase strategy. Initially, it presents a generalized schema to the LLMs to establish a foundational understanding. Subsequently, it employs targeted prompting to elicit detailed associations between query phrases and specific database entities, thereby enhancing accuracy in mapping queries to database structures without incurring excessive token costs. ODIS [63] proposes a SimSQL method to retrieve additional knowledge from cross-domain databases. This method utilizes the BM25 algorithm to measure the resemblance in SQL keywords and schema tokens. The top examples from each database are selected as the demonstrations that closely align with target SQL.

While retrieval methods have improved the efficiency and effectiveness of additional information acquisition, the associated increase in computational cost cannot be ignored. Reducing the computational cost of selecting and embedding additional information remains a significant challenge. Moreover, current studies typically use domain-specific knowledge in the form of text sentences as the main additional information, with insufficient exploration of formulated or structured domain knowledge bases. Further integration of multiple forms of additional information can enhance the performance of NL2SQL models on more domain-specific databases.

V. NL2SQL Translation Methods

In this section, we delve into NL2SQL translation methods using language models. As shown in Figure 6, we will elaborate on their encoding (Section V-A), decoding (Section V-B), and task-specific prompt strategies (Section V-C). We will then discuss how intermediate representations can benefit the NL2SQL translation process (Section V-D).

A. Encoding Strategy

Encoding in the NL2SQL task refers to the process of transforming NL and database schema into a structured format that can be effectively utilized by a language model. This transformation is crucial as it converts unstructured and semi-structured data into a form that can be processed for generating SQL queries. The encoding process involves capturing the semantic meaning of the NL input and the structural information of the database schema, enabling the model to understand and map the user's intent to the corresponding SQL query.

There are three primary encoding strategies in NL2SQL models, as shown in Figure 7, each with its unique approach to transforming NL and database schema: *1) sequential encoding*, *2) graph-based encoding*, and *3) separate encoding*.

1) Sequential Encoding Strategy: Sequential encoding is one of the primary strategies used in NL2SQL models, where both the NL and the database schema are treated as sequences of tokens. The key idea is to linearize the input data, enabling sequence-based models to capture semantic and syntactic information effectively. For example, several works [61], [71] employ the T5 model [43] to encode the input NL and database schema in order. This line of work includes T5+NatSQL+Token Preprocessing [61], T5+PICARD [71], and others.

BRIDGE [77] enhances the alignment between text and the database schema by representing the NL question and database schema as a tagged sequence and inserting matched database cell values (anchor texts) after the corresponding fields. Similarly, N-best List Rerankers [69] appends database content to column names to enhance the model's serialization scheme. RESDSQL [7] uses a ranking-enhanced encoder to sort and filter schema items, thereby reducing the complexity of schema linking during encoding. This method ensures that the most relevant schema items are prioritized, improving the overall efficiency of the encoding process. CatSQL [65] utilizes the pre-trained GraPPa encoding network [36] to concatenate the NL, database schema, and additional information into a single input sequence, generating hidden state sequences. This approach integrates multiple sources of information, enhancing the model's ability to capture complex relationships.

Although LLM-based NL2SQL methods do not explicitly employ an input encoding strategy, most of the LLMs rely on process input sequences through the self-attention mechanism. This mechanism can also be considered a form of the encoding process, generating the next word in a sequence based on the preceding words. Therefore, in this line of research, MAC-SQL [64], SuperSQL [46], C3-SQL [60], DAIL-SQL [6], DIN-SQL [5], PET-SQL [55], CodeS [47], and DTS-SQL [53] employ the sequential encoding strategy.

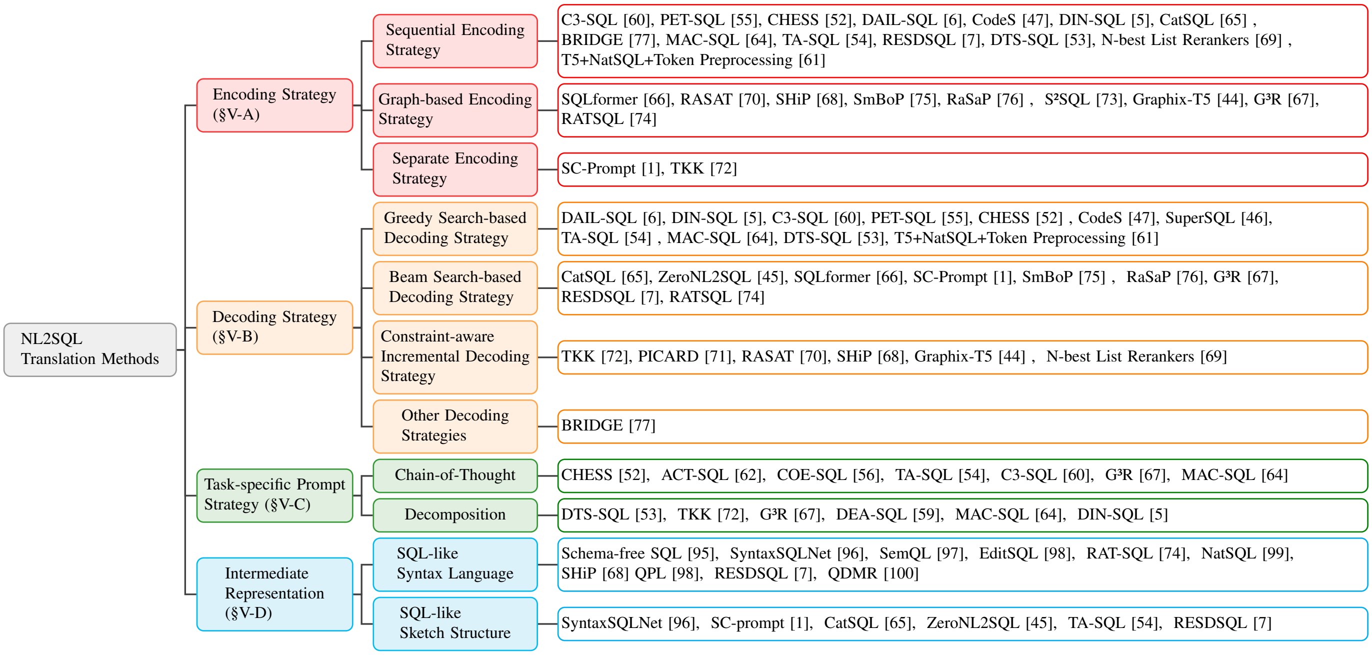
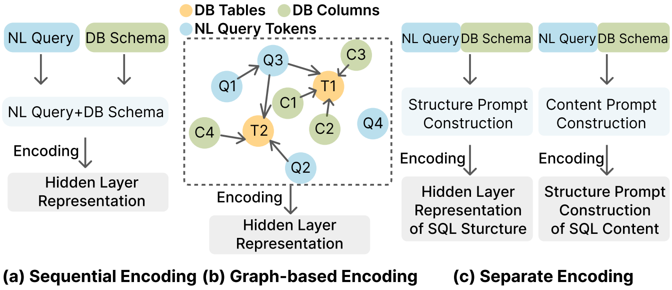

NL2SQL Translation Methods
Encoding Strategy (§V-A)
Sequential Encoding Strategy
C3-SQL [60], PET-SQL [55], CHESS [52], DAIL-SQL [6], CodeS [47], DIN-SQL [5], CatSQL [65] , BRIDGE [77], MAC-SQL [64], TA-SQL [54], RESDSQL [7], DTS-SQL [53], N-best List Rerankers [69] , T5+NatSQL+Token Preprocessing [61]
Graph-based Encoding Strategy
SQLformer [66], RASAT [70], SHiP [68], SmBoP [75], RaSaP [76] , S²SQL [73], Graphix-T5 [44], G³R [67], RATSQL [74]
Separate Encoding Strategy
SC-Prompt [1], TKK [72]
Decoding Strategy (§V-B)
Greedy Search-based Decoding Strategy
DAIL-SQL [6], DIN-SQL [5], C3-SQL [60], PET-SQL [55], CHESS [52] , CodeS [47], SuperSQL [46], TA-SQL [54] , MAC-SQL [64], DTS-SQL [53], T5+NatSQL+Token Preprocessing [61]
Beam Search-based Decoding Strategy
CatSQL [65], ZeroNL2SQL [45], SQLformer [66], SC-Prompt [1], SmBoP [75] , RaSaP [76], G³R [67], RESDSQL [7], RATSQL [74]
Constraint-aware Incremental Decoding Strategy
TKK [72], PICARD [71], RASAT [70], SHiP [68], Graphix-T5 [44] , N-best List Rerankers [69]
Other Decoding Strategies
BRIDGE [77]
Task-specific Prompt Strategy (§V-C)
Chain-of-Thought
CHESS [52], ACT-SQL [62], COE-SQL [56], TA-SQL [54], C3-SQL [60], G³R [67], MAC-SQL [64]
Decomposition
DTS-SQL [53], TKK [72], G³R [67], DEA-SQL [59], MAC-SQL [64], DIN-SQL [5]
Intermediate Representation (§V-D)
SQL-like Syntax Language
Schema-free SQL [95], SyntaxSQLNet [96], SemQL [97], EditSQL [98], RAT-SQL [74], NatSQL [99], SHiP [68] QPL [98], RESDSQL [7], QDMR [100]
SQL-like Sketch Structure
SyntaxSQLNet [96], SC-prompt [1], CatSQL [65], ZeroNL2SQL [45], TA-SQL [54], RESDSQL [7]

Fig. 6: A Taxonomy of NL2SQL Translation Methods based on their Design Choices.


DB Tables
DB Columns
NL Query Tokens
NL Query
DB Schema
NL Query+DB Schema
Encoding
Hidden Layer Representation
Q1
Q2
Q3
Q4
T1
T2
C1
C2
C3
C4
Encoding
Hidden Layer Representation
NL Query
DB Schema
NL Query
DB Schema
Structure Prompt Construction
Content Prompt Construction
Encoding
Encoding
Hidden Layer Representation of SQL Sturcture
Structure Prompt Construction of SQL Content

(a) Sequential Encoding (b) Graph-based Encoding (c) Separate Encoding

Fig. 7: An Overview of the Encoding Strategies.

The sequential encoding strategy is simple and intuitive, but its linear approach, which treats all inputs as sequences of tokens, may fail to fully capture the complex relationships between the database schema and NL query. This can affect the model's ability to understand and generate complex queries.

2) Graph-based Encoding Strategy: Graph-based Encoding is a sophisticated strategy used in NL2SQL models where both the NL and the database schema are represented as graphs. This approach leverages the inherent relational structure of databases and the complex inter-dependencies in the input data [36], [44], [66], [68], [70], [73]–[76]. Compared to sequential encoding methods, graph-based encoding approaches preserve the original database schema's topology, providing a richer context for each element, which can enhance the performance of the model in generating accurate SQL queries.

RAT-SQL [74] introduces a relation-aware self-attention mechanism, allowing the model to explicitly consider and utilize predefined relational information when jointly encoding the question and the database schema. These relationships are represented as a graph structure, and through this graph-based encoding, RAT-SQL can more effectively capture the structural information in the schema and its alignment with the NL query. Building on the RAT-SQL encoder, SmBoP+GraPPa [75] employs the GraPPa [36] pre-trained model to jointly encode the NL query and database schema. Similarly, RaSaP [76] combines the ELECTRA pre-training model and enhances joint encoding effectiveness based on the RAT-SQL encoder.

Using ELECTRA as its pre-trained language model, S^{2}SQL [73] enhances encoding by injecting syntactic structure information at the encoding stage, improving the semantic understanding and generation of models. G^{3}R [67] introduces the LGESQL encoder, which captures and integrates multi-source heterogeneous information by constructing and utilizing a heterogeneous graph and a Graph Attention Network (GAT) [101], thereby enhancing the representation capability for NL and database schema. RASAT [70] augments the self-attention mechanism in the T5 model with relation-aware self-attention, which can incorporate a variety of relational structures while inheriting pre-trained parameters from the T5 model [43]. SHiP [68] leverages strong typing encoding and key relationship encoding to enhance the representation capability of the input information during the encoding phase. SQLformer [66] introduces learnable table and column embeddings to select the most relevant tables and columns in the context of the database schema. The Graphix-T5 model [44] introduces graph-aware layers allowing the model to incorporate structural information directly into the encoding process, significantly improves the model's ability to understand and generate SQL queries that correctly reflect the structure of the database, and achieves better results on multiple NL2SQL benchmarks.

The graph-based encoding strategy can effectively express the complex relationships between input data, enabling the model to more accurately understand and generate complex SQL queries involving multiple relationships and conditions. However, this approach requires more sophisticated algorithms

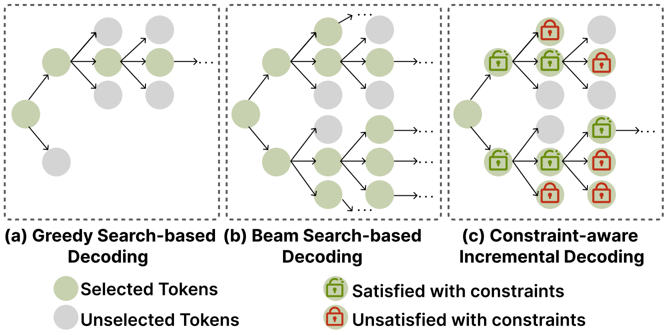

(a) Greedy Search-based Decoding
(b) Beam Search-based Decoding
(c) Constraint-aware Incremental Decoding
Selected Tokens
Unselected Tokens
Satisfied with constraints
Unsatisfied with constraints

Fig. 8: An Overview of the Decoding Strategies.

for constructing and processing graph structures and may need a large amount of training data to fully leverage its structural advantages, making it potentially less suitable for scenarios with limited training data.

3) Separate Encoding Strategy: Separate encoding strategy is an approach in NL2SQL models where different parts of the NL (such as clauses and conditions) are encoded separately and then combined to form the final SQL. These separate encodings are then combined at a later stage to generate the final SQL.

Early works encode the NL query and database schema separately, such as SQLNet [41] and Seq2SQL [40]. The main reason for using separate encoding in early models was the format mismatch between NL and database schema, which necessitated separate encoding. This approach is not conducive to schema linking and is therefore rarely used nowadays. However, separate encoding allows for specialized processing of different types of input data, leveraging the strengths of various encoding techniques.

Based on the pre-trained T5 model, TKK [72] employs task decomposition and multi-task learning strategies in encoding by breaking down the complex NL2SQL task into multiple subtasks and progressively acquiring and combining knowledge. Similarly, SC-Prompt [1] divides text encoding into two stages: the structure stage and the content stage, with each stage being encoded separately.

While the separate encoding strategy requires multiple processing of input data, which may extend the training and inference time of the model, it allows for more refined handling and understanding of different aspects of queries. This provides the model with the flexibility to handle various query tasks and allows for the adjustment of encoding strategies at different stages based on the needs of the task, thereby enhancing overall performance.

B. Decoding Strategy

Decoding plays a crucial role in NL2SQL translation, as it is responsible for converting the representations generated by the encoder into the target SQL queries. The choice of decoding strategy directly affects the quality and performance of the generated SQL queries. An excellent decoding strategy not only produces syntactically correct SQL queries but also ensures that the semantics of the SQL queries align with the NL and can even optimize the execution efficiency of the queries.

We will introduce several key decoding strategies employed by existing NL2SQL models, as shown in Figure 8, namely: *1) greedy search-based decoding strategy*, *2) beam search-based decoding strategy*, and *3) constraint-aware incremental decoding strategy*. We will then discuss the advantages and disadvantages of each approach.

1) Greedy Search-based Decoding Strategy: The greedy search-based decoding strategy is a simple and fast approach for decoding. At each decoding step, greedy search selects the token with the highest current probability as the output. This strategy builds the final output sequence by continuously choosing the locally optimal solution, as shown in Figure 8(a).

Since the default decoding strategy of GPT series models (*e.g.,* GPT-4) is greedy search-based decoding, NL2SQL solutions based on GPT fall into this category. These include MAC-SQL [64], SuperSQL [46], C3 [60], DAIL-SQL [6], DIN-SQL [5], and PET-SQL [55].

Besides GPT-based models, CodeS [47] is a series of NL2SQL models incrementally pre-trained on StarCoder [49] using a large SQL-related corpus, which is also a decoder-only model using greedy search-based decoding as its decoding strategy. DTS-SQL [53] with DeepSeek LLM [102] also uses greedy search for decoding. Some early NL2SQL models also use greedy search as their decoding strategy. For example, SQLNet [41] and Seq2SQL [40] both use greedy search-based decoding for generating SQL queries.

Greedy search is widely used in many NL2SQL models due to its fast decoding speed and simple implementation. However, greedy search only considers the optimal solution at the current step, ignoring long-term dependencies and overall optimal solutions, which may result in the generation of SQL queries that are not globally optimal. Moreover, errors at each step could lead to biases in decision-making in subsequent steps, potentially accumulating errors throughout the decoding process, especially when handling complex queries.

2) Beam Search-based Decoding Strategy: Beam search is a decoding strategy designed to explore a larger search space and thus tends to achieve better results. During the decoding process, beam search retains multiple candidate sequences (*i.e.,* “beams”) instead of selecting only one optimal token at each step. In other words, beam search selects the top-k candidate tokens with the highest probabilities and expands them until a complete output sequence is generated at each decoding step, as shown in Figure 8(b).

Given the advantages of beam search, several NL2SQL models employ this decoding strategy [1], [7], [45], [65], [74]–[76], [98]. For example, RAT-SQL [74] combines relation-aware graph structure encoding and generation techniques. During the decoding process, RAT-SQL uses beam search to generate multiple candidate SQL queries, which are then reranked, and the optimal query is selected based on graph structure information. Unlike RAT-SQL, EditSQL [98] employs a context encoding strategy, incorporating dialogue history information into the model. During the decoding process, it uses the beam search-based decoding strategy to generate candidate SQL queries and utilizes dialogue context information to select and optimize the queries.

SmBoP [75] uses a semi-autoregressive bottom-up decoding approach, improving decoding efficiency by parallelizing the construction and scoring of sub-trees, achieving logarithmic time complexity. Based on SmBoP, RaSaP [76] employs a semi-autoregressive bottom-up semantic parser combined with a tree contextualization mechanism, enhancing the accuracy and efficiency of generating SQL queries. SQLformer [66] also uses an autoregressive decoder to generate SQL queries, based on the node type, adjacency, and the previous action in the SQL syntax tree.

SC-Prompt [1] designs a keyword-constrained decoding to ensure the validity of generated SQL structures and a structure-guided decoding to guarantee the model fills the correct content. G^{3}R [67] proposes the AST-Grammar bipartite graph and knowledge-enhanced re-ranking mechanism in the decoding stage, dynamically modeling the AST and grammar rules for generating SQL queries. CatSQL [65] proposes Column Action Templates (CAT) to efficiently fill multiple SQL clause slots through a parameter-sharing decoding network, combined with semantic correction techniques to ensure the generated SQL queries are semantically correct.

RESDSQL [7] employs a skeleton-aware decoder that first generates the SQL skeleton and then the actual SQL query, thereby implicitly guiding the SQL parsing process. Unlike RESDSQL, ZeroNL2SQL [45] uses beam search to retain the top-k hypotheses as candidate sets in the SQL sketch generation stage, which are then used for subsequent SQL query completion and predicate calibration processes.

Compared to greedy search, beam search-based decoding retains multiple candidate sequences during the decoding process, which means that more data needs to be processed at each decoding step. This significantly increases the demand for memory and computational resources and results in slower decoding speeds than greedy search. However, by considering the multiple most probable candidates at each decoding step, beam search achieves a broader search space and is more likely to generate more accurate and complex SQL queries.

3) Constraint-aware Incremental Decoding Strategy: The constraint-aware incremental decoding strategy, introduced by PICARD [71] (Parsing Incrementally for Constrained Auto-Regressive Decoding), is specifically designed for NL2SQL tasks. This strategy aims to ensure the generation of syntactically correct SQL queries by incorporating constraints during the decoding process, as shown in Figure 8(c).

Roughly speaking, NL2SQL solutions that employ the constraint-aware incremental decoding strategy generate SQL queries incrementally while adhering to SQL grammar rules. Specifically, at each step of the decoding process, the model not only predicts the next token but also applies a set of constraints to validate the syntactic correctness of the partial SQL query generated so far. These constraints are derived from SQL grammar rules, which help the model avoid generating invalid SQL structures. Therefore, PICARD [71] ensures that each token added to the query adheres to the required SQL syntax. This method reduces the likelihood of generating erroneous or incomplete SQL queries, thereby improving the overall accuracy and reliability of the NL2SQL translation.

Due to its ability to improve the accuracy of SQL generation, PICARD [71] has been adopted as a decoding strategy by several models, including TKK [72], RASAT [70], T5 [71], SHiP [68], N-best List Rerankers [69], and Graphix-T5 [44]. Compared to versions without PICARD, these models have shown improved accuracy.

The constraint-aware incremental decoding strategy, through incremental decoding and progressively adding constraints, may require more computational resources and processing time. However, it effectively ensures the syntactic correctness of the generated SQL queries, reducing the production of erroneous and invalid queries, which is suitable for generating structurally complex SQL queries.

4) Other Decoding Strategies: In addition to the three common decoding strategies mentioned above, some models have proposed special decoding strategies. To improve the decoding accuracy, BRIDGE [77] introduces some simple heuristic rules to prune the search space of the sequence decoder, proposing Schema-Consistency Guided Decoding to ensure that the generated SQL queries are consistent with the database schema. This strategy continuously checks whether the generated SQL queries match the database schema during the decoding process and adjusts the decoding path based on the matching results.

C. Task-specific Prompt Strategy

In the era of LLMs, prompt engineering can harness the capabilities of LLMs and has been widely adopted in natural language processing, with various frameworks developed for specific tasks [103]–[105]. In the NL2SQL field, task-specific prompt strategy refers to the tailored prompt engineering techniques used in the NL2SQL translation process. These strategies instruct the LLMs to optimize the SQL query generation process according to task-specific rules, improving the accuracy of translating complex semantic NL query into the corresponding SQL query.

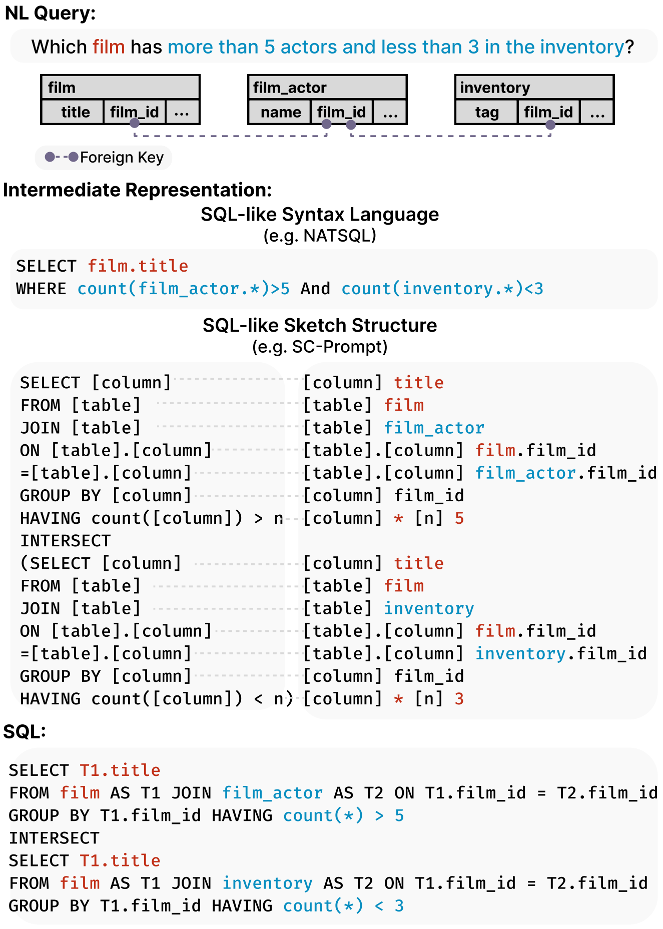
Roughly speaking, there are two kinds of task-specific prompt strategies used by NL2SQL solutions, namely:

- *Chain-of-Thought*: this strategy drives the model to think and reason about the task goal sequentially.
- *Decomposition*: this strategy breaks down the final task and uses different modules to reason about subtasks separately.

1) Chain-of-Thought: Chain-of-Thought (CoT) [106], widely recognized for its effectiveness, facilitates the integration of specialized semantic rules into the generation process of language models. This strategy fully shows the thinking and reasoning process of LLMs, enhancing the accuracy and interpretability of the generated results. The application of CoT in NL2SQL tasks not only enhances the performance of NL2SQL models but also ensures that the SQL statements generated are more aligned with human expectations [107]. With further research, CoT is often used as a key component in subsequent work rather than a separate trick. For example, CHESS [52] transforms NL into SQL statements using a streamlined pipeline that relies on LLMs and CoT. This process comprises entity and context retrieval, schema selection, SQL generation, and revision.

In addition, the integration of CoT with other techniques can significantly enhance the performance of NL2SQL models. These techniques comprises in-context learning [56], [62], logical synthesis [54], calibration with hints [60], [67] and multi-agent system [64]. In-context learning enriches the CoT by embedding a deeper comprehension of the linguistic environment, facilitating a more precise semantic mapping to SQL constructs. The application of logical synthesis ensures that the inferential steps within the CoT are not only contextually grounded but also adhere to formal logical structures, thereby enhancing the validity of the deductive process. Calibration with hints serves to fine-tune the model's responses, aligning them with the intricacies of NL, which is critical for accurately capturing user intent in the translation from NL to SQL. Moreover, the integration of multi-agent systems and CoT introduces a distributed problem-solving approach, where specialized agents collaborate to address various stages of the CoT, such as schema linking and generation. This collaborative framework not only expedites the CoT process but also enhances the system's resilience and adaptability. In essence, these techniques contribute to a more sophisticated and robust NL2SQL framework, characterized by a heightened degree of precision and reliability in the translation of complex NL into accurate SQL.

2) Decomposition: The decomposition strategy decomposes a given target into several sub-tasks and then addresses the sub-tasks step by step to achieve improved generation results. There are differences in the granularity of subtask decomposition in different works. Compared with CoT, the decomposition strategy can make each sub-module focus on executing a sub-step of the generation process, so as to ensure better generation quality and interpretability. DTS-SQL [53] splits the work task into two subtasks, schema linking, and generation, to close the performance gap between open-source LLMs and closed-source LLMs. More works apply decomposition strategies to smaller fine-grained processes. The TKK framework [72] divides the initial NL2SQL parsing tasks into various small individual subtasks, with each corresponding to the mapping of the NL query to one or more clauses of the SQL query. This decomposition approach allows the model to concentrate on learning the generation of each clause, thereby compelling the model to understand the problem, the database schema, and the alignment between each SQL clause. Similar task decomposition strategies also exist in the work of G^{3}R [67] and DEA-SQL [59]. Moreover, the decomposition of the NL can also significantly alleviate the complexity involved in model learning. MAC-SQL [64] incorporates a Decomposer agent designed to break down the user's original problem into several subproblems. This decomposition process aims to lessen the complexity of the origin question, enabling the generation of simpler SQL queries to solve each individual subproblem. DIN-SQL [5] employs a sophisticated categorization module for decomposition. It classifies queries into distinct complexity groups: EASY, NON-NESTED, and NESTED, with the reference of NL and database schema. This module is fundamental for the subsequent decomposition process, which meticulously dissects complex queries into simpler sub-problems. By strategically identifying and separating schema linking, join conditions and nested structures, the module facilitates a structured generation of SQL queries and amplifies the accuracy of translating complex the NL query into executable SQL.



```
SELECT film.title
WHERE count(film_actor.*)>5 And count(inventory.*)<3
```

```
SELECT [column]                 [column] title
FROM [table]                    [table] film
JOIN [table]                    [table] film_actor
ON [table].[column]             [table].[column] film.film_id
=[table].[column]               [table].[column] film_actor.film_id
GROUP BY [column]               [column] film_id
HAVING count([column]) > n      [column] * [n] 5
INTERSECT
(SELECT [column]                [column] title
FROM [table]                    [table] film
JOIN [table]                    [table] inventory
ON [table].[column]             [table].[column] film.film_id
=[table].[column]               [table].[column] inventory.film_id
GROUP BY [column]               [column] film_id
HAVING count([column]) < n)     [column] * [n] 3
```

```
SELECT T1.title
FROM film AS T1 JOIN film_actor AS T2 ON T1.film_id = T2.film_id
GROUP BY T1.film_id HAVING count(*) > 5
INTERSECT
SELECT T1.title
FROM film AS T1 JOIN inventory AS T2 ON T1.film_id = T2.film_id
GROUP BY T1.film_id HAVING count(*) < 3
```

Fig. 9: An Example of the Intermediate Representation.

In general, the decomposition strategy can decompose the NL2SQL translation process into several sub-tasks, which allows the sub-modules in each stage to focus on improving the quality of the corresponding generation, thus optimizing the final SQL query generation. However, this strategy also brings the challenge of increased computational cost, which increases the difficulty of model training and deployment

D. Intermediate Representation for NL2SQL Translation

As mentioned before, the NL2SQL task is challenging due to the complexity and ambiguity of NL queries, as well as the formal and structured nature of SQL. Thus, researchers try to simplify this process by designing a *grammar-free* intermediate representation compared to SQL as the bridge between the "free-form" NL query and the "constrained and formal" SQL query. Roughly speaking, an intermediate representation (IR) is a structured yet flexible grammar that captures the essential components and relationships of an NL query without the strict syntax rules of SQL.

As shown in Figure 9, we categorize existing intermediate representation strategies into two groups: *1) SQL-like syntax* *language*, and *2) SQL-like sketch structure*. SQL-like syntax language strategies transform the NL query into short SQL-like expressions, bridging the gap between the NL query and the final SQL. SQL-like sketch structure strategies construct the sketch rules and maps the NL into an SQL-like skeleton, which can simplify the parsing process and improve the generation quality of the final SQL query.

1) SQL-like syntax language: The SQL-like syntax language approach typically involves transforming a user's query into an intermediate SQL-like expression, which bridges the gap between NL and predefined rules. This process serves to alleviate the challenges associated with semantic parsing and query generation. The early approach to intermediate representation involved leveraging information retrieval techniques to translate the original question and schema information into an intermediate syntax representation, which was then converted into SQL [95], [108]. Subsequent research efforts have focused on consolidating or eliminating partial clauses or operations in SQL queries to simplify SQL-like syntax language. In the research of Schema-free SQL [95], the original question can be transformed into an intermediate representation even in the absence of user knowledge about schema information. SyntaxSQLNet [96] removes portions of the `FROM` and `JOIN` clauses in the syntax language, while SemQL [97] removes the `FROM`, `JOIN`, `ON` and `GROUP BY` clauses and combines `WHERE` and `HAVING` conditions. EditSQL [98] adds `WHERE` and `HAVING` conditions but retains the `GROUP BY` clause.

Recent research has made advancements in simplifying the structure and generality of syntax languages in order to enhance their parsing efficiency. For example, Natural SQL (NatSQL) [99] is a widely recognized SQL-like syntax language that eliminates SQL statement operators, keywords, set operators, and other elements seldom found in user problem descriptions. It enhances schema linking by minimizing the necessary number of schema items. The combination of NatSQL with pre-trained language models is able to achieve effective generation results on existing benchmarks [7], [61]. The Query Plan Language (QPL) [109] leverages the problem decomposition strategy to improve the parsing of intricate SQL queries. By breaking down a SQL query into modularized sub-queries, the complexity of the original query is reduced. This approach mitigates parsing difficulties associated with complex problems and cross-domain complex queries. Question Decomposition Meaning Representation (QDMR) [100], [110] decomposes the original question into a number of atomic questions. Each atomic question serves as an intermediate representation of the original question and can be translated into a set of small-scale formal operations involving tasks such as selecting entities, retrieving attributes, or aggregating information. The QDMR pipeline is structured to be executed sequentially in order to provide a comprehensive answer to the original question.

The utilization of SQL-like syntax language has shown the potential to bridge the gap between user queries and databases. However, there are still some limitations in this area. Previous studies have been limited by high complexity and inadequate coverage of database structures [99]. With the increase in database size and domain knowledge, the simplicity of SQL-like syntax language faces great challenges. Furthermore, some existing SQL-like syntax languages still require manual construction and adjustment, which increases the cost and difficulty of deploying the model. Current research efforts aim to enhance the quality of SQL generation by integrating SQL-like syntax language with other generation methodologies.

2) SQL-like sketch structure: With reference to the structural characteristics of SQL, researchers design the sketch rules that mirror SQL structure for parsing, enabling the mapping of various NL into the sketch space of rules. This kind of approach reduces the complexity of content parsing and enhances the quality of generation.

Early works typically parse questions into SQL-like sketch structures using fixed SQL rules and neural networks. The SyntaxSQLNet [96] encompasses the SQL-based syntax tree alongside its corresponding decoder. This model breaks down the decoding process into nine modules, predicting various operators, keywords, and database information individually. Subsequently, it combines these predicted tokens to perform the decoding process. Similarly, Lee introduces a self-attention-based database schema encoder designed for decoding SQL clauses [111]. The model incorporates several SQL sketch-specific decoders, with each decoder comprising a collection of submodules defined by the SQL syntax of each clause. These initial approaches are constrained by the base model's performance and do not achieve relatively good results in large-scale benchmarks, but they do provide a reference for subsequent research.

In recent years, the development of language models has allowed researchers to design more elaborate SQL-like structures for parsing. The SC-prompt [1] utilizes a two-stage divide and conquer method for NL2SQL parsing. During the initial phase, it instructs PLM to generate specific SQL structures, such as query commands and operators, while also supplying placeholders for any missing identifiers. In the subsequent phase, it directs the PLM to generate SQL structures containing actual values to fill the previously provided placeholders. CatSQL [65] constructs a template sketch with slots serving as initial placeholders. Different from the former, this sketch is much more general. Its base model can focus on the parsing of user queries to fill these placeholders, consequently decreasing the computational resource cost. Furthermore, it implements a novel semantic correction algorithm to assess the semantic accuracy of the resulting SQL queries and rectify any semantic issues detected in the generated queries. ZeroNL2SQL [45] integrates the schema alignment capabilities of PLM with the complex reasoning capabilities of LLMs. Initially, it utilizes PLM to produce SQL sketches for achieving schema alignment and subsequently employs LLMs to execute complex content reasoning for populating missing information. Additionally, it also proposes a predicate calibration method for guiding the design of language models for SQL sketches based on database instances and selecting the optimal SQL query. TA-SQL [54] combines pandas code and symbolic representation to generate an abstract sketch of SQL and uses this sketch to align with schema information in subsequent modules to generate complete SQL.

Moreover, several recent works cover both SQL-like syntax language and SQL-like sketch transition methods. For instance, RESDSQL [7] introduces a rank-enhanced encoding and skeleton-aware decoding framework, which separates schema linking from skeleton parsing. During the decoding generation phase, its decoder initially produces the SQL skeleton and then generates the actual SQL query. This approach implicitly constrains the SQL parsing and governs the quality of generation. When combined with NatSQL, RESDSQL demonstrates the ability to further enhance the quality of SQL query generation.

In general, SQL-like sketch structure can be more easily combined with other strategies, such as decomposition strategy or SQL-like syntax language strategy. In addition, it can more fully utilize the comprehension and cloze capabilities of existing LLMs and reduce the dependence on professionals. With the expansion of database size and domain-specific knowledge bases, the design of intermediate representation should be more flexible and generalized, adapting to the NL2SQL translation process in the case of complex conditions and multiple nested queries.

VI. Post-Processing Strategies for NL2SQL

After the NL2SQL model generates the SQL, we can refine it to better meet user expectations. This process is called post-processing. Post-processing involves adding information or using other models to further improve the generated SQL. The current approach to post-processing primarily focuses on correction, consistency, execution, and the n-best list to amend the SQL. It is worth noting that the NL2SQL model may adopt multiple post-processing methods for better results.

A. SQL Correction Strategies

The SQL generated by the model may sometimes contain syntax errors. SQL Correction Strategies aim to prompt LLMs to correct syntax errors in SQL.

SQL queries generated by NL2SQL models may lack or contain redundant keywords, such as `DESC`, `DISTINCT`, and `Aggregate` functions. To address these issues, DIN-SQL [5] proposes a self-correction module that guides the model to correct these errors. This module is implemented in the zero-shot setting, where the model is only provided with the buggy SQL and asked to fix the errors. The study suggests two different prompts for different models: a general prompt for the CodeX model and a mild prompt for the GPT-4 model. In the general prompt, the model is requested to identify and correct errors in the "BUGGY SQL". The mild prompt does not assume the SQL query has errors. Instead, it asks the model to provide potential issues with clauses.

Given the SQL predicates ($column^0$, $value^0$) predicted by LLMs, there might be two types of errors: incorrectly predicted values and incorrect predicted columns. ZeroNL2SQL [45] adopts a multi-level matching approach that incrementally expands the matching scope across three levels (columns, tables, and databases) to sequentially match predicate values. The matched predicate values are then returned to the LLMs, helping it generate SQL queries consistent with the database content.

In the multi-agent collaboration framework, MAC-SQL [64] designs a Refiner agent, whose primary function is to detect and correct SQL errors. After receiving an SQL query, the Refiner Agent will diagnose the SQL statement from three aspects: syntactic correctness, execution feasibility, and whether it retrieves non-empty results from the database. If the check fails, it will reason based on the original SQL and error feedback information or modification guidance signals to correct the erroneous SQL statement. The core function is to enable the model to perform self-diagnosis and self-correction, thereby enhancing the robustness and accuracy of the overall framework.

Current research [45] primarily focuses on specific syntax errors in generated SQL, developing a specialized correction strategy to address these errors. However, as the capabilities of LLMs continue to improve, syntax errors in generated SQL will decrease, and guiding LLMs to automatically correct potential SQL errors may become more general [5], [64].

B. Output Consistency

To enhance the consistency of model outputs, the concept of self-consistency [112] has been proposed in previous work. The motivation behind the self-consistency method is that, in complex reasoning problems, there are multiple reasoning paths that can lead to a unique correct answer. It first samples multiple different reasoning paths and then selects the most consistent answer to significantly improve the quality of the output. The NL2SQL task, being similar to reasoning tasks, can also have multiple ways to write SQL queries that express the same meaning.

The work of C3-SQL [60] incorporates the Consistency Output (CO) component, which aims to maintain the consistency of the generated SQL queries by overcoming the inherent randomness and uncertainty in the outputs of large language models, thereby improving zero-shot NL2SQL performance. Specifically, CO first samples multiple reasoning paths to generate different SQL answers. Then, these SQL queries are executed on the database, and the execution results are collected. After removing errors from all results, a voting mechanism is applied to these execution results to determine the most consistent SQL as the final SQL. This method enables models to leverage the collective knowledge derived from these multiple paths, resulting in more reliable outcomes in generating SQL queries.

DAIL-SQL [6] also mentions integration with the self-consistency module, achieving a 0.4% improvement compared to not having this module, although it notes that equipping this module would significantly increase costs and time. Compared to C3-SQL [60], CHESS [52] also reduces noise in model outputs by selecting the most consistent output from three SQL samples.

The existing self-consistency method generates different results by increasing the randomness of LLMs' outputs at high temperatures and then uses majority voting to select the final result. However, there are reports [113] that high temperatures may reduce the performance of LLM because of increasing model hallucinations and that the diversity of a single model is usually insufficient. To address this issue, PET-SQL [55] proposes the cross-consistency strategy, which instructs multiple LLMs to generate SQL at lower temperatures and then votes on the execution results of the SQL. This cross-consistency strategy not only diversifies SQL queries but also maintains the performance of LLMs at low-temperature settings.

The above methods can ensure a certain degree of consistency in the queries generated multiple times by a single or multiple LLMs, reducing the effects caused by model randomness. This can improve the accuracy of the generated SQL to some extent, but it also significantly increases inference cost and time.

C. Execution-Guided Strategies

In the NL2SQL scenario, the execution result of an SQL query is also a crucial piece of information, as it may indicate the quality of the query itself. For instance, if the query execution results in errors or $NULL$ values, it might indicate an issue with the SQL query. Currently, many models incorporate the results of SQL queries into their post-processing steps, using these results to guide subsequent processing of the SQL. The current NL2SQL models have varying approaches to utilizing the output results.

ZeroNL2SQL [45] continuously generates SQL queries through an executable check process after obtaining multiple candidate SQL sketches. It feeds back error messages to the LLMs to achieve an executable query. This process repeats continuously until the feedback limit p is reached, after that the current sketch will be discarded, and the next candidate SQL sketch is selected. This approach allows the model to enable the LLMs to complete the SQL query and select the optimal query as the final output.

Compared to ZeroNL2SQL, CodeS [47] can directly output complete SQL statements. During the beam search process, the model generates four SQL candidates and selects the first executable one as the final SQL result.

In MAC-SQL [64], a specialized Refiner Agent is used to detect and automatically correct SQL errors. The Refiner can use external tools to execute SQL, obtain feedback, and optimize any incorrect SQL queries.

To reflect human behavior when writing complex SQL queries, CHESS [52] returns not only the database schema, question, and candidate SQL queries but also the execution results of the SQL queries to LLMs. Specifically, CHESS starts with a draft query and refines it based on its execution results, making necessary adjustments to the SQL query in case of syntax errors.

Execution-Guided Strategies can refine SQL based on its execution results, ensuring that the generated SQL can retrieve data from the database. However, this approach significantly increases the SQL generation time when dealing with large databases.

D. N-best Rerankers Strategies

In NL2SQL tasks, particularly in cross-domain scenarios, the generated SQL queries may exhibit nuanced variances in both structure and semantics. The N-best list reranking method involves reordering the top n results from the original model, often using a larger model or incorporating additional knowledge sources. On the Spider dataset, fine-tuning a BERT model as a reranker, as demonstrated in Bertrand-dr [114], has been shown to successfully improve multiple NL2SQL models.

However, the improvement from reordering algorithms in Bertrand-dr [114] can be unstable and highly dependent on threshold settings. In some threshold configurations, it can even produce negative effects. G^{3}R [67] proposes a feature-enhanced reranker based on PLM to address the shortcomings of Bertrand-dr [114]. The SQL reranker leverages a PLM with hybrid prompt tuning to integrate into the PLM's knowledge, effectively bridging gaps between various domains without adding extra parameters. Contrastive learning is then used to push away the representation distance of candidate queries, making them more distinguishable [115].

Zeng et al. [69] proposes two rerankers from the perspectives of consistency and correctness. To improve consistency, query plans generated by independent models can be used to explore N-best reranking. Then, to enhance correctness, they introduced a heuristic algorithm that applies schema linking on the N-best list to impose constraints missing in PICARD. The combined reranking method produces improvements in T5 models.

Similarly, ReFSQL [116] employs a ranking algorithm to retrieve the most closely related generated results from the retriever and generator module. In addition, the change of ranking candidates can further improve the quality of the final answer generation. ZeroNL2SQL [45] consists of ranking the candidate set of SQL sketches. It leverages the Question-Aware Aligner to identify the candidate that most effectively aligns with the user's question intent and specific requirements. The system then calculates an alignment score between the candidate SQL sketch and the user question to ensure a more semantic match with the user's query.

The N-best reranking method can further filter generated candidate SQL s based on additional knowledge or larger models and is widely used in PLM-based methods. However, this approach is less commonly used with LLMs that have more parameters and stronger inference capabilities.

VII. NL2SQL Benchmarks

In this section, we will first elaborate on the different types of NL2SQL datasets, highlighting their characteristics, as shown in Figure 10. We will then perform an in-depth analysis of existing NL2SQL datasets (Section VII-I).

An Overview of NL2SQL Benchmarks. With the advancement of the NL2SQL field, various datasets have been introduced to address the diverse challenges of NL2SQL tasks.

As shown in Figure 10, early NL2SQL datasets were single-domain with simple SQL queries and databases. As the field progressed, cross-domain datasets merged, presenting challenges for the generalization capabilities of NL2SQL systems. The introduction of multi-turn datasets required NL2SQL systems to infer target SQL queries based on contextual understanding. Subsequently, multilingual challenges

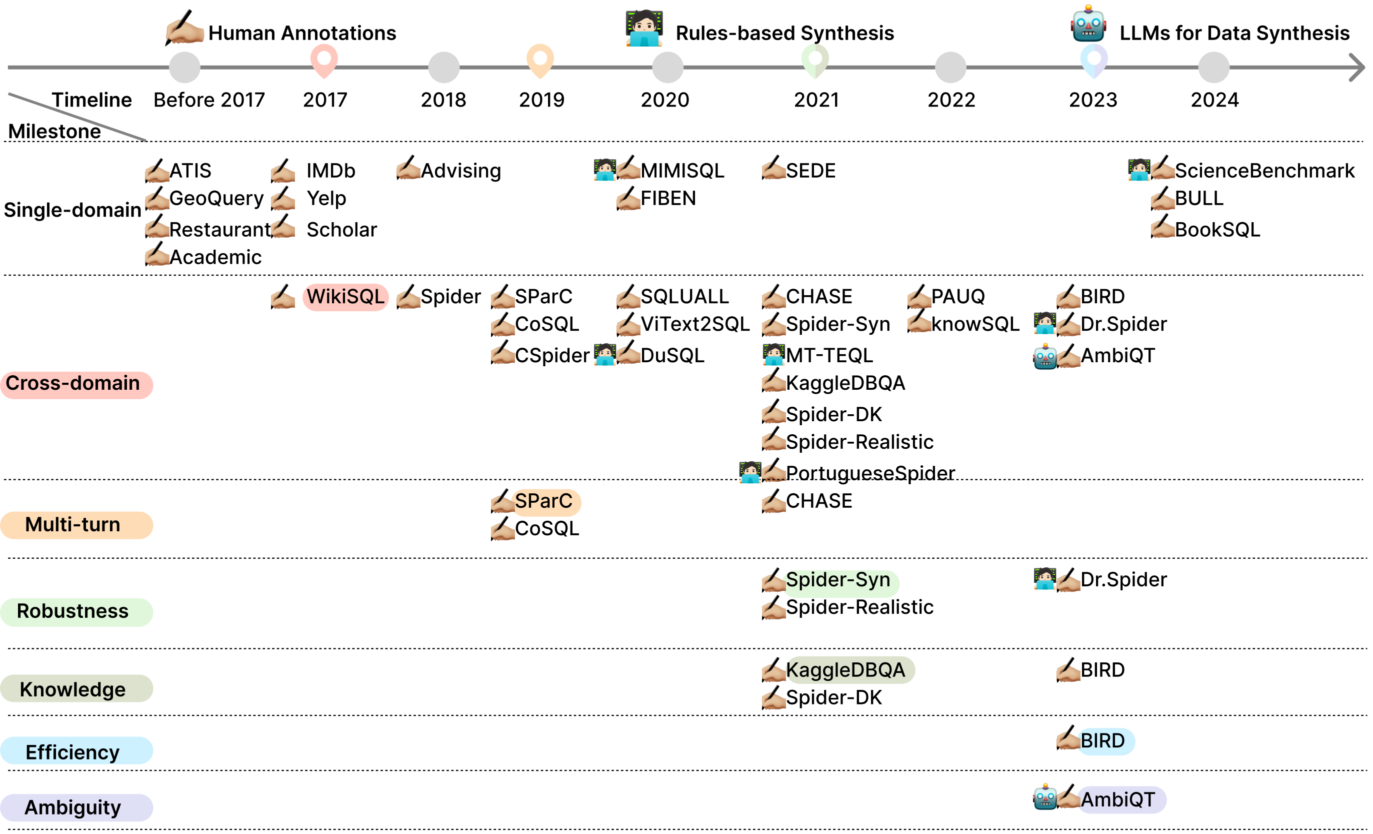


Fig. 10: Timeline for NL2SQL Benchmarks.

TABLE II: Statistics of NL2SQL Benchmarks.

Dataset	Redundancy Measure			DB Complexity					Query Complexity				
	#-Questions	#-Unique Queries	#-Questions / #-Queries	#-DBs	#-Tables	#-Tables / DB	#-Cols / Table	#-Records / DB	Tables	Selects	Agg	Scalar Func	Math Comp
ATIS [117]	5280	947	5.6	1	25	25	5.24	162243	8.39	1.79	0.22	0	0
GeoQuery [118]	877	246	3.6	1	7	7	4.14	937	2.22	2.19	0.92	0	0.01
Restaurants [119]	378	23	16.4	1	3	3	4.00	19295	2.43	1.17	0.35	0	0
Academic [108]	196	185	1.1	1	17	17	3.12	58249674	3.48	1.04	0.54	0	0
IMDb [120]	131	89	1.5	1	17	17	3.94	40147386	2.91	1.01	0.30	0	0
Yelp [120]	128	110	1.2	1	8	8	5	4823945	2.41	1	0.45	0	0
Scholar [121]	817	193	4.2	1	10	10	2.50	147416275	3.38	1.02	0.68	0	0.02
WikiSQL [40]	80654	80257	1	26531	26531	1	6.34	17	1	1	0.28	0	0
Advising [122]	4387	205	21.4	1	15	15	7.40	332596	3.41	1.21	0.40	0	0.11
Spider [31]	11840	6448	1.8	206	1056	5.13	5.01	8980	1.83	1.17	0.54	0	0
SParC [123]	10228	8981	1.1	166	876	5.28	5.14	9665	1.58	1.10	0.44	0	0
CoSQL [124]	8350	8007	1	166	876	5.28	5.14	9665	1.54	1.11	0.42	0	0
CSpider [125]	11840	6408	1.8	206	1056	5.13	5.01	8980	1.83	1.17	0.54	0	0
MIMICSQL [126]	20000	10000	2	-	-	-	-	-	1.74	1	0.84	0	0
SQUALL [127]	11276	8296	1.4	2108	4028	1.91	9.18	71	1.22	1.29	0.40	0.03	0.16
FIBEN [128]	300	233	1.3	1	152	152	1	11668125	5.59	1.56	0.97	0	0.04
ViText2SQL [129]	9693	5223	1.9	166	876	5.28	5.14	9665	1.17	1.12	0.54	0	0
DuSQL [130]	25003	20308	1.2	208	840	4.04	5.29	20	1.49	1.25	0.73	0	0.30
PortugueseSpider [131]	9693	5275	1.8	166	876	5.28	5.14	9665	1.85	1.17	0.54	0	0
CHASE [132]	15408	13900	1.1	350	1609	4.60	5.19	4594	1.81	1.16	0.31	0	0
Spider-Syn [133]	1034	550	1.9	166	876	5.28	5.14	9665	1.68	1.17	0.59	0	0
Spider-DK [134]	535	283	1.9	169	887	5.25	5.14	9494	1.71	1.16	0.54	0	0
Spider-Realistic [135]	508	290	1.8	166	876	5.28	5.14	9665	1.79	1.21	0.50	0	0
KaggleDBQA [136]	272	249	1.1	8	17	2.12	10.53	595075	1.25	1.05	0.69	0	0.04
SEDE [137]	12023	11421	1.1	1	29	29	7.28	-	1.90	1.29	0.94	0.49	0.49
MT-TEQL [138]	489076	4525	108.1	489076	3279004	6.70	5.51	-	1.69	1.15	0.53	0	0
PAUQ [139]	9876	5497	1.8	166	876	5.28	5.14	9693	1.82	1.17	0.53	0	0
knowSQL [140]	28468	-	-	488	-	-	-	-	-	-	-	-	-
Dr.Spider [141]	15269	3847	4	549	2197	4	5.54	28460	1.81	1.19	0.52	0	0
BIRD [50]	10962	10840	1	80	611	7.64	7.14	4585335	2.07	1.09	0.61	0.20	0.27
AmbiQT [142]	3046	3128	1	166	876	5.28	5.14	9665	1.85	1.17	0.51	0	0.01
ScienceBenchmark [143]	5031	3654	1.4	-	-	-	-	-	1.45	1	0.24	0	0.07
BULL [48]	7932	5864	1.4	3	78	26	14.96	85631	1.22	1	0.18	0.42	0.05
BookSQL [144]	78433	39530	2	1	7	7	8.86	1012948	1.25	1.12	0.78	0.39	0.22

were introduced which contained NL questions and databases in different languages, thus enhancing the complexity of multilingual comprehension for NL2SQL systems. With the growing demand for NL2SQL applications in real-world scenarios, researchers have shifted focus towards the robustness of NL2SQL systems, leading to the development of datasets designed to validate robustness from multiple perspectives. The most recent advancements have introduced large datasets targeting specific real-world domains, further promoting the application and development of NL2SQL systems in practice. The advancement of NL2SQL datasets not only highlights the continuous progress in the NL2SQL field but also reflects the new challenges that arise.

A. Single-Domain NL2SQL Datasets

Early NL2SQL datasets focused on specific domains, often featuring simple SQL query structures. The ATIS [117] dataset, for example, is centered on flight information, while GeoQuery [118] uses a database of US geographical facts. Other datasets like Restaurants [119], Academic [108], IMDb [120], Yelp [120], and Scholar [121] are tailored to their respective domains, as indicated by their names. Recently, several large single-domain datasets such as MIMICSQL [126], FIBEN [128], SEDE [137], ScienceBenchmark [143], BULL [48] and BookSQL [144] have been introduced. Unlike the earlier single-domain datasets, these datasets feature more complex databases and SQL queries specific to particular scenarios. This advancement indicates a growing focus on evaluating the performance and practical application of NL2SQL systems within specific domains.

B. Cross-Domain NL2SQL Datasets

After the introduction of early single-domain datasets, the focus of the NL2SQL field shifted towards cross-domain datasets. Cross-domain datasets challenge the generalization capabilities of NL2SQL systems across different domains, requiring these systems to generalize not only to new SQL queries but also to new databases. The first cross-domain dataset introduced was WikiSQL [40], which extracted tables from Wikipedia across various domains. It includes 80,654 manually written NL question and SQL query pairs, each pair corresponding to a unique table. Following WikiSQL, the Spider [31] dataset was introduced, featuring 11,840 NL questions and 6,448 unique SQL queries over 200 databases spanning 138 different domains. Unlike WikiSQL, each (NL, SQL) pair in Spider corresponds to a database with multiple complex relational tables rather than a single table.

More recently, the BIRD [50] dataset has gained popularity as a cross-domain dataset. Compared to Spider, BIRD further increases the complexity of SQL queries by including SQL functions and operations not present in Spider, thus providing an even greater challenge for NL2SQL systems.

C. Multi-Turn NL2SQL Datasets

As the NL2SQL field progressed, there has been a growing interest in building NL2SQL systems centered on multi-turn interactive dialogues. To meet this demand, various multi-turn datasets have been developed. SParC [123] is a multi-turn, cross-domain dataset that features approximately 4.3K question sequences, cumulatively forming over 12K (NL, SQL) pairs. It evolved from the Spider dataset and derived from controlled user interactions. Each question sequence in SParC contains contextually continuous (NL, SQL) pairs, requiring the NL2SQL system to understand and maintain the context throughout the interactions. Additionally, CoSQL [124] employs a Wizard-of-Oz setup for data collection, resulting in over 30K turns. Compared to SParC, CoSQL introduces additional challenges such as unanswerable user questions, further testing the context understanding of NL2SQL systems. CHASE [132] is the first multi-turn Chinese dataset. Compared to SParC and CoSQL, CHASE further reduces the number of context-independent questions and easy SQL queries, pushing the boundaries in the dialogue-centered NL2SQL systems.

D. NL2SQL Datasets: Considering Robustness

In real-world applications, NL2SQL systems need to handle diverse user groups and various database domains, making robustness a growing focus within the community. Spider-Syn [133] is a dataset derived from the Spider development set through manual annotation. It simulates real-world scenarios where users may not be familiar with the database schema by replacing schema-related words in the original NL questions with synonyms. Similarly, Spider-Realistic [135] is a subset selected from the Spider development set, consisting of 508 complex examples with equivalently paraphrased NL questions. Further, Dr.Spider [141] dataset applies 17 different perturbations to the databases, NL questions, and SQL queries from Spider, providing a comprehensive evaluation of NL2SQL robustness. Compared to Spider-Syn and Spider-Realistic, Dr.Spider includes not only semantic-preserving perturbations but also semantic-changing perturbations.

E. NL2SQL Datasets: Considering SQL Efficiency

Databases in real-world scenarios often contain massive amounts of data, and a user question can be solved by multiple SQL queries. However, these SQL queries can vary in execution efficiency, which has attracted attention from the community. BIRD [50] is the first cross-domain dataset to incorporate SQL efficiency, containing 95 large databases with 33.4GB in total size. As shown in Table II, compared to the Spider dataset, which has approximately 9K records per database, BIRD has over 4,500K records per database. Additionally, BIRD introduces a metric for evaluating SQL execution efficiency called the Valid Efficiency Score (VES), which will be further discussed in Section VIII.

F. Knowledge-Augmented NL2SQL Datasets

NL2SQL systems often require domain-specific knowledge to effectively perform the NL2SQL task in real-world applications within specific domains. Integrating domain-specific knowledge into the NL2SQL task has recently gained increasing interest. KaggleDBQA [136] includes database documentation, such as column and table descriptions and categorical value descriptions, providing NL2SQL systems with rich domain knowledge. In addition, Spider-DK [134], based on the Spider development set, defines and adds five types of domain knowledge to the NL questions, focusing on evaluating the ability to understand and utilize domain knowledge. Similarly, BIRD [50] also includes expert-annotated domain knowledge, which can be further categorized into four main types: numeric reasoning knowledge, domain knowledge, synonym knowledge, and value illustration.

G. NL2SQL Datasets: Considering NL Ambiguity

In real-world NL2SQL tasks, various ambiguities often arise, such as semantic ambiguities in NL and overlapping database schema. Evaluating the performance under the ambiguity of NL2SQL systems has gained growing attention. AmbiQT [142] is the first dataset proposed to evaluate the coverage of ambiguity, which contains over 3000 examples spanning four different types of ambiguities. In AmbiQT, each NL question corresponds to two valid SQL queries due to the specific ambiguity.

H. Synthetic NL2SQL Datasets

As shown in Table II, despite the fact that many datasets have been manually annotated so far, the rapid development of LLMs in the NL2SQL field has led to a growing demand for NL2SQL data. Consequently, the automatic NL2SQL data synthesis has become an area of increasing interest within the community. MIMICSQL [126] uses a template-based method to generate template questions and the corresponding SQL queries. However, the template questions may be unreasonable or unnatural. Therefore, manual annotation is employed to rewrite the template questions into NL questions. Similarly, ScienceBenchmark [143] uses a template-based method to initially generate SQL queries. However, it differs by employing the GPT-3 model to perform the SQL-to-NL translation. MT-TEQL [138] is auto-generated by applying a comprehensive set of metamorphic relations, which are semantics-preserving linguistic and schema transformations, to the Spider development set. Additionally, both Dr.Spider [141] and AmbiQT [142] utilize the GPT-3.5 model to assist in dataset creation. Specifically, Dr. Spider focuses on paraphrasing NL questions, while AmbiQT creates synonyms for the NL questions.

I. In-depth Analysis of Existing NL2SQL Datasets

To further explore and compare the complexity of different datasets, we utilize NL2SQL360 [46] evaluation framework to statistically analyze each dataset. The statistics include three main categories: Redundancy Measure, DB Complexity, and Query Complexity. The detailed results are shown in Table II. Redundancy Measure includes three aspects: the number of questions, the number of queries, and their ratio. DB Complexity encompasses four aspects: total number of databases, total number of tables, average number of tables per database, average number of columns per table, and average number of records per database. Query Complexity measures the average number of tables, `SELECT` keyword, aggregate functions (*e.g.,* SUM), scalar functions (*e.g.,* ROUND), and mathematical computations (*e.g.,* "+") in each SQL query. Notably, for datasets without publicly available dev/test datasets, such as CHASE [132], BULL [48], and BookSQL [144], we only include statistics for the public dataset splits. Additionally, for datasets that do not have publicly available data, such as knowSQL [140], the corresponding values in Table II are indicated with a symbol of "–".

From the Redundancy Measure perspective, we observe a trend from early datasets to recent ones where datasets have grown in size, including increases in the number of questions and unique queries. Specifically, MT-TEQL [138] stands out with the highest number of NL questions and the largest ratio of NL questions to SQL queries, due to its automated transformation of NL questions, generating a large volume of variants.

From the Database Complexity perspective, the number of databases (and tables) in datasets correlates with the tasks they serve. Single-domain datasets like BookSQL [144] typically have fewer databases, while datasets focused on robustness evaluation tasks like Dr.Spider [141] and MT-TEQL [138] feature a higher number of databases. FIBEN [128] has the highest average number of tables per database (152) and large database content (approximately 11.67M records per database).

Regarding Query Complexity, datasets like FIBEN [128] and SEDE [137] exhibit SQL queries with numerous tables and aggregate functions, reflecting characteristics found in real-world financial domains and Stack Exchange websites. Additionally, recent datasets show a growing emphasis on Scalar Functions and Mathematical Computations in SQL queries, which introduces challenges in SQL generation structure not seen in earlier datasets.

Discussion. Despite the growing number of datasets proposed by the NL2SQL community, we find that current datasets still exhibit a gap in SQL complexity compared to real-world scenarios. For example, recent datasets generally feature lower counts of `SELECT` keyword, indicating fewer nested SQL queries or complex set operations. Moreover, challenges related to Scalar Functions and Mathematical Computations also need further attention. We look forward to the community proposing more datasets that focus on these challenges in the future.

VIII. Evaluation and Error Analysis

In this section, we will first introduce the well-established metrics for evaluating NL2SQL solutions (Section VIII-A). We will then present existing evaluation toolkits for assessing NL2SQL solutions from different perspectives with low human cost (Section VIII-B). Finally, we will introduce an error taxonomy to organize and analyze the SQL errors from the NL2SQL process (Section VIII-C).

A. Evaluation Metrics

Evaluation metrics are crucial for assessing the performance of NL2SQL systems. Researchers have proposed various metrics to evaluate from multiple perspectives. We define N as the dataset size, Q_i as the NL question of the i-th example, V_i as the execution result set of the ground-truth SQL query Y_i

and $\hat{V}_i$ as the execution result set of the predicted SQL query $\hat{Y}_i$ by the NL2SQL system.

Execution Accuracy (EX). Execution Accuracy (EX) [31] evaluates the performance of the NL2SQL system by comparing whether the execution result sets of the ground-truth SQL queries and the predicted SQL queries are identical. It can be computed by:

$$EX = \frac{\sum_{i=1}^{N} \mathbb{1}(V_i = \hat{V}_i)}{N}, \tag{1}$$

where $\mathbb{1}(\cdot)$ is an indicator function that equals 1 if the condition inside is satisfied, and 0 otherwise. False negatives could occur because different SQL queries corresponding to semantically different NL questions may produce identical execution result sets (*e.g.,* when the `WHERE` clause in SQL query does not effectively filter the database values).

String-Match Accuracy (SM). String-Match Accuracy (SM) (also called Logical Form Accuracy) [145] simply compares whether the ground-truth SQL query and the predicted SQL query are identical as strings. It may penalize SQL queries that produce the correct execution result sets but do not have the exact string match with the ground-truth SQL queries. It can be computed as follows:

$$SM = \frac{\sum_{i=1}^{N} \mathbb{1}(Y_i = \hat{Y}_i)}{N}. \tag{2}$$

Component-Match Accuracy (CM). Component-Match Accuracy (CM) [31] evaluates the detailed performance of the NL2SQL system by measuring the exact matching of different SQL components such as `SELECT`, `WHERE` and others between the ground-truth SQL query and the predicted SQL query. For a specific SQL component C. The computation can be formalized as follows:

$$CM^C = \frac{\sum_{i=1}^{N} \mathbb{1}(Y_i^C = \hat{Y}_i^C)}{N}, \tag{3}$$

where Y_i^C is the component of SQL query Y_i. To correctly determine if an SQL component matches, some SQL components (*e.g.,*`WHERE`) do not consider order constraints. Furthermore, the F1 score can be calculated to report the overall performance of each component.

Exact-Match Accuracy (EM). Exact-Match Accuracy [31] is based on the Component-Match Accuracy (CM) and measures whether all SQL components $\mathbb{C} = \{C_k\}$ of the predicted SQL query match the ground-truth SQL query. It can be computed as follows:

$$EM = \frac{\sum_{i=1}^{N} \mathbb{1}(\bigwedge_{C_k \in \mathbb{C}} Y_i^{C_k} = \hat{Y}_i^{C_k})}{N}. \tag{4}$$

Valid Efficiency Score (VES). Valid Efficiency Score (VES) [50] measures the execution efficiency of valid SQL queries. It considers both the accuracy and efficiency of SQL execution, which can be computed as follows:

$$VES = \frac{\sum_{i=1}^{N} \mathbb{1}(V_i = \hat{V}_i) \cdot \mathbf{R}(Y_i, \hat{Y}_i)}{N}, \tag{5}$$

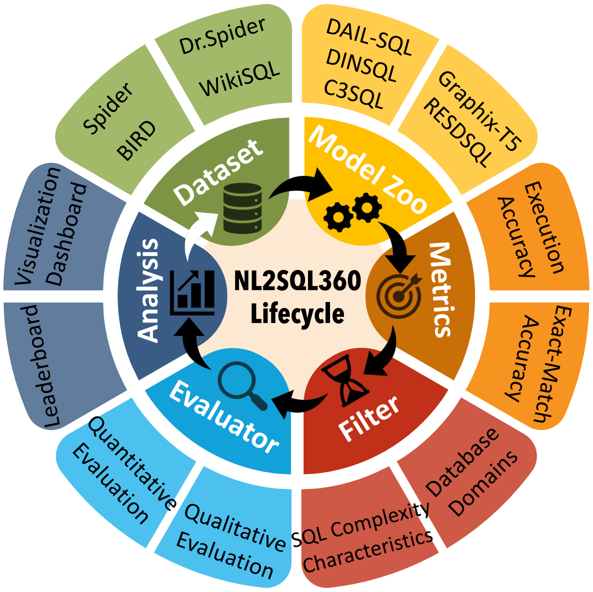


Fig. 11: An Overview of NL2SQL360 [46].

$$\mathbf{R}(Y_i, \hat{Y}_i) = \sqrt{\frac{\mathbf{E}(Y_i)}{\mathbf{E}(\hat{Y}_i)}}, \tag{6}$$

where $\mathbf{R}(\cdot)$ measures the relative execution efficiency of the predicted SQL query compared to the ground-truth SQL query, eliminating uncertainties due to machine status. $\mathbb{E}(\cdot)$ measures the efficiency of specific SQL query, which can be refer to execution time, memory usage and more.

Query Variance Testing (QVT). Query Variance Testing (QVT) [46] measures the robustness and flexibility of the NL2SQL system in handling variations in NL questions. For a given SQL query Y_i, there typically exist multiple corresponding NL questions, denoted as pairs $\{(Q_1, Y_i), (Q_2, Y_i), \dots, (Q_m, Y_i)\}$. QVT metric can be calculated by:

$$QVT = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\sum_{j=1}^{m_i} \mathbb{1}\left(\mathcal{F}(Q_{ij}) = Y_i\right)}{m_i} \right), \tag{7}$$

where m_i denotes the number of different NL variations for the SQL query Y_i, and $\mathcal{F}(Q_{ij})$ is the predicted SQL query for the j-th NL variation of Y_i. It is notable that QVT only considers (NL, SQL) pairs that the NL2SQL system can correctly process at least one NL variation. This implicitly constructs a specific test dataset for each NL2SQL system.

B. NL2SQL Evaluation Toolkits

With the advancement of the NL2SQL field, the performance of NL2SQL systems on various benchmark datasets has improved. However, in real-world applications, the style of NL questions, the database schemas across different domains, and the characteristics of SQL queries may vary in different scenarios. Evaluating NL2SQL systems using overall metrics on existing benchmark hold-out datasets cannot fully capture their real-world performance and robustness.

To provide a comprehensive understanding of NL2SQL systems in real-world scenarios, recent advancements have been made in NL2SQL evaluation toolkit [46], [138].

MT-TEQL [138] is a unified framework for evaluating the performance of NL2SQL systems in handling real-world variations in NL questions and database schemas. It is based on a metamorphic testing approach, implementing semantic-preserving transformations of NL questions and database schemas to automatically generate their variants without manual efforts. Specifically, it includes four types of transformations for NL questions: *Prefix Insertion*, *Prefix Insertion*, *Prefix Substitution* and *Synonym Substitution*. It also includes eight types of transformations for database schemas: *Normalization*, *Flattening*, *Opaque Key*, *Table Shuffle*, *Column Shuffle*, *Column Removal*, *Column Renaming*, and *Column Insertion*.

A recent advancement is NL2SQL360 [46], a multi-angle evaluation framework that provides fine-grained assessments of NL2SQL systems in various specific scenarios. Compared to MT-TEQL, it focuses on the varied characteristics of SQL queries across various application scenarios. For example, in Business Intelligence (BI) scenarios, SQL queries typically include aggregate functions, nested queries, or top-k queries. As shown in Figure 11, NL2SQL360 comprises six core components: *Dataset*, *Model Zoo*, *Metrics*, *Dataset Filter*, *Evaluator*, and *Analysis*, which provide a unified interface that is agnostic to specific models and datasets. Users can use public or private datasets to automatically conduct systematic evaluations of NL2SQL systems across various metrics. They can also customize metrics based on specific scenario requirements, or analyze the performance on subsets with scenario-related SQL characteristics. This provides researchers and practitioners with deep insights into the performance of NL2SQL systems in specific application scenarios.

Despite the progress made by the community in developing NL2SQL evaluation toolkits, real-world NL2SQL systems still face numerous variations that differ from existing benchmark datasets, such as database schema design principles. We anticipate the development of more powerful evaluation toolkits in the future to enhance the application of NL2SQL systems in practical scenarios.

C. A Taxonomy for NL2SQL Errors Analysis

Error Analysis refers to the process of examining the errors made by a model aimed at directing future corrective measures and enhancing model performance. Error Analysis is a critical aspect of NL2SQL research, as it provides valuable insights into the limitations and challenges faced by current models. By systematically examining errors, researchers can identify specific areas for improvement, enhance model robustness, and develop more effective training strategies.

In this section, we first introduce the existing NL2SQL error taxonomy. We then call for a standardized and effective taxonomy for NL2SQL Errors. Finally, we proposed design principles and tried to design a two-level taxonomy.

Existing Taxonomies for NL2SQL Errors Analysis. In existing sophisticated NL2SQL research [5], [52], [146]–[148], error analysis has begun to be conducted, and various taxonomies of error types have been proposed.

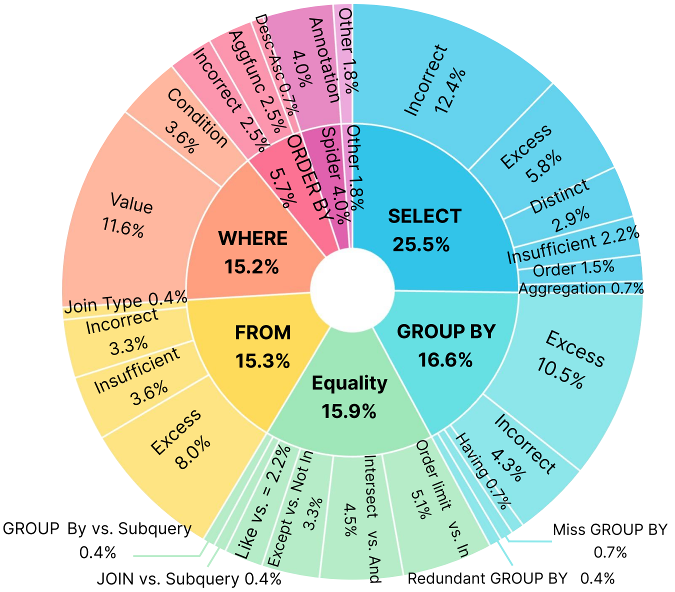


Fig. 12: Statistics of Din-SQL Errors by Our Taxonomy.

Ning et al. [147] developed a detailed taxonomy of error types. They categorized the error types along two dimensions: (1) *the syntactic dimension* shows the specific parts of the SQL where an error occurs, organized by SQL keywords like `WHERE` and `JOIN`; (2) *the semantic dimension* highlights which aspects of the natural language description are misinterpreted by the model, such as misunderstanding a value or the name of a table. The combination of these two dimensions results in a total of 40 distinct categories.

DIN-SQL [5] manually examined the failed queries and classified them into six categories: (1) *Schema Linking*, where the model fails to identify column names, table names, or entities mentioned in the questions; (2) *JOIN*, where the model fails to identify all the necessary tables or the correct foreign keys for joins; (3) *GROUP BY*, where the SQL requires a `GROUP BY`, but the model either does not recognize the need for `GROUP BY` or chooses incorrect columns for `GROUP BY`; (4) *Queries with Nesting and Set Operations*, where the model fails to recognize or correctly implement the nested structure or set operation; (5) *Invalid SQL*, where the generated SQL contains syntax errors; and (6) *Miscellaneous*, which includes cases that do not fit into the other categories.

SQL-PLAM [148] categorizes errors into five types: (1) *Schema Linking*, where the model fails to select the relevant tables required for NL; (2) *Misunderstanding Database Content*, where the model is not able to accurately interpret the data within the tables; (3) *Misunderstanding Knowledge Evidence*, where the model fails to use or ignores human-annotated evidence; (4) *Reasoning*, which includes an inadequate understanding of the question, leading to queries that lack the necessary reasoning steps to generate correct results; and (5) *Syntax-Related* Errors, where the generated SQL contains syntax errors.

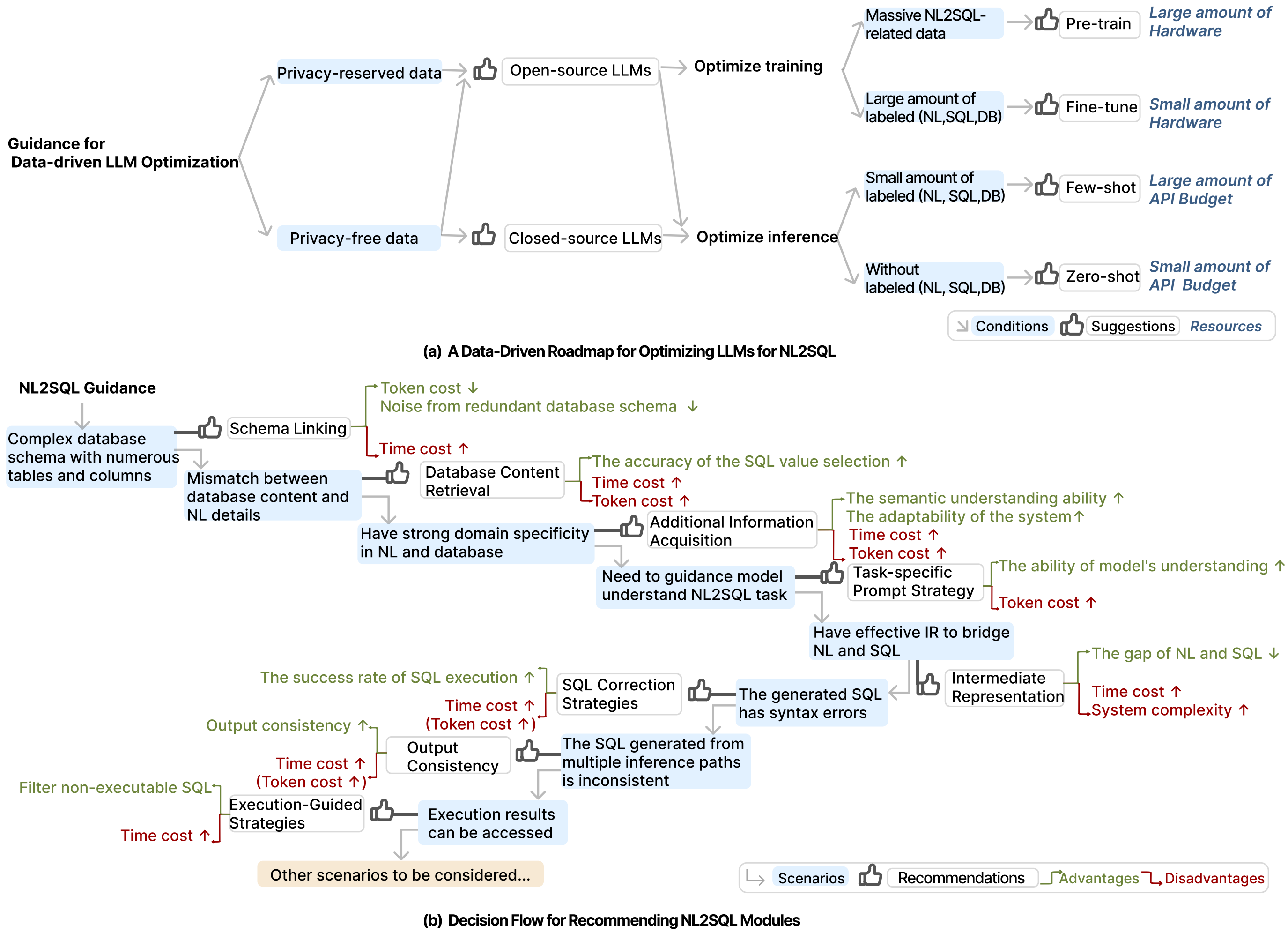
Our Taxonomy for NL2SQL Errors Analysis. The above taxonomy is designed for specific datasets and has limited scalability. Due to inconsistent taxonomies, the annotated data cannot be shared or compared between different studies. The inconsistency of error taxonomies hinders collaborative improvements and the cumulative advancement of knowledge in NL2SQL Error Analysis. To effectively address this issue, it is imperative to develop a standardized and effective taxonomy of error types. We propose the following principles to guide the development of NL2SQL error taxonomy:

- *Comprehensiveness*: The taxonomy should encompass all potential errors that could occur during the NL2SQL conversion process.
- *Mutual Exclusivity*: Each error type should be clearly distinct with no overlap, to avoid ambiguity in error classification.
- *Extensibility*: The taxonomy should be adaptable to incorporate new error types as NL2SQL technologies and methodologies evolve.
- *Practicality*: The taxonomy should be practical and applicable in real-world settings, aiding developers in diagnosing and correcting errors effectively.

Following these principles, we attempted to design a taxonomy containing two levels:

- *Error Localization*: This level focuses on identifying the specific parts of the SQL where errors occur, such as in the `SELECT` clause. It is vital for precisely locating where misunderstandings or misinterpretations arise, thereby facilitating targeted corrections.
- *Cause of Error*: This level focuses on understanding why the model is wrong when generating SQL. For example, value errors in the `WHERE` clause may indicate the model's insufficient ability to understand and retrieve database content. On the other hand, conditional errors in the `WHERE` clause typically reveal flaws in semantic understanding, where the model fails to grasp the logical requirements of the query.

A Case Study of Error Analysis. We collected the errors generated by DIN-SQL [5] on the Spider [31] and manually classified them according to the taxonomy we designed, as illustrated in Figure 12. The results indicate that this taxonomy is effective; however, we believe that the NL2SQL Error Taxonomy is a field that requires ongoing exploration and updates, and it is impossible to design a perfect taxonomy in


Guidance for
Data-driven LLM Optimization
Privacy-reserved data
Privacy-free data
Open-source LLMs
Closed-source LLMs
Optimize training
Optimize inference
Massive NL2SQL-related data
Pre-train
Large amount of Hardware
Large amount of labeled (NL,SQL,DB)
Fine-tune
Small amount of Hardware
Small amount of labeled (NL, SQL,DB)
Few-shot
Large amount of API Budget
Without labeled (NL, SQL,DB)
Zero-shot
Small amount of API Budget
Conditions
Suggestions
Resources

(a) A Data-Driven Roadmap for Optimizing LLMs for NL2SQL

NL2SQL Guidance
Complex database schema with numerous tables and columns
Schema Linking
Token cost ↓
Noise from redundant database schema ↓
Time cost ↑
Mismatch between database content and NL details
Database Content Retrieval
The accuracy of the SQL value selection ↑
Time cost ↑
Token cost ↑
Have strong domain specificity in NL and database
Additional Information Acquisition
The semantic understanding ability ↑
The adaptability of the system ↑
Time cost ↑
Token cost ↑
Need to guidance model understand NL2SQL task
Task-specific Prompt Strategy
The ability of model's understanding ↑
Token cost ↑
Have effective IR to bridge NL and SQL
Intermediate Representation
The gap of NL and SQL ↓
Time cost ↑
System complexity ↑
The generated SQL has syntax errors
SQL Correction Strategies
The success rate of SQL execution ↑
Time cost ↑
(Token cost ↑)
The SQL generated from multiple inference paths is inconsistent
Output Consistency
Output consistency ↑
Time cost ↑
(Token cost ↑)
Execution results can be accessed
Execution-Guided Strategies
Filter non-executable SQL
Time cost ↑
Other scenarios to be considered...
Scenarios
Recommendations
Advantages
Disadvantages

(b) Decision Flow for Recommending NL2SQL Modules

Fig. 13: A Data-Driven Roadmap and a Decision Flow for Recommending NL2SQL modules.

one attempt. Therefore, we hope the research community will continuously engage in refining and enhancing this taxonomy.

IX. Practical Guidance for Developing NL2SQL Solutions

In this section, we present a roadmap for optimizing LLMs for the NL2SQL task, based on data privacy and data volume (Section IX-A). Designing an NL2SQL system is complex and necessitates careful consideration of various scenarios. Therefore, we provide a decision flow for selecting appropriate NL2SQL modules tailored to various scenarios (Section IX-B).

A. A Data-Driven Roadmap of Optimizing LLMs for NL2SQL

In Figure 13(a), we outline a strategic roadmap designed to optimize LLMs for NL2SQL task, based on data privacy and data volume. Data privacy affects the choice of open-source and closed-source LLMs, while data volume affects the strategies for optimization for training and inference.

Condition 1: Data Privacy. When dealing with privacy-reserved data, the only option is to select open-source LLMs because closed-source models typically require the invocation of external APIs, which may involve sending data to servers and potentially lead to data breaches. The advantage of open-source models lies in their ability to allow users to entirely control the training and inference process in the local environment, thereby enhancing data privacy protection.

Condition 2: Data Volume. For open-source LLMs, it is possible to optimize both the training and inference phases. However, for closed-source LLMs, due to the opaque nature of the models, optimization can only be applied to the inference stage. When massive amounts of NL2SQL task-related data are available (often in the tens of gigabytes), LLMs can be optimized by pre-training. When hundreds to thousands of (NL, SQL, DB) pairs are available, the model can be optimized by fine-tuning. For scenarios with limited labeled data, a few-shot learning approach is viable. In the absence of labeled data, zero-shot methods become particularly important. Note that consideration must also be given to the corresponding hardware resources and API costs when adopting different optimization strategies.

The insights and strategies discussed for optimizing LLMs for the NL2SQL task can be applied to all NLP tasks. This universality underscores the importance of carefully considering data privacy and volume in all NLP applications.

B. Decision Flow of Selecting NL2SQL Modules

In Figure 13(b), we outline recommendations for selecting different NL2SQL modules for various scenarios and discuss the corresponding positive and negative impacts.

Scenario 1: *Complex Database Schema with Numerous Tables and Columns.* In this scenario, we recommend the adoption of Schema Linking strategies. This approach primarily benefits the system by reducing token costs and decreasing noise from the redundant database schema. However, it is important to note that these advantages come at the expense of increased time costs.

Scenario 2: *Mismatch between Database Content and NL Details.* In this scenario, we recommend the adoption of Database Content Retrieval strategies. This approach primarily benefits the system by increasing the accuracy of SQL value selection. However, this leads to increased time and token costs.

Scenario 3: *Execution Results Can be Accessed.* In this scenario, we recommend the adoption of Execution-Guided Strategies. This approach primarily benefits the system by filtering non-executable SQL. However, the drawback is the increased time cost due to the need to execute queries on the database to obtain results. This time cost can be significantly high, especially when dealing with very large databases.

In summary, while each module offers distinct advantages for handling specific NL2SQL scenarios, it is crucial to balance these advantages against the associated disadvantages in the NL2SQL system design.

X. Open Problems

In this section, we present several open problems and discuss the research challenges for NL2SQL.

A. Open NL2SQL Problem

In many real-world scenarios, such as government open data platforms, citizens may pose questions that require querying multiple databases and aggregating results for a response. For example, a citizen might ask, “What is the average processing time for tax returns in the last five years?” Answering this question involves retrieving relevant tables from different databases, such as tax records, processing time logs, and statistical reports databases, and then generating multiple SQL queries over these databases.

The Open NL2SQL problem needs to translate NL queries into SQL queries by not only relying on a fixed database but also identifying multiple relevant databases (tables) within an enterprise-scale DBMS. In addition, one open NL query may have multiple corresponding SQL queries, where each SQL query accesses different databases. This approach contrasts with the traditional NL2SQL task (*i.e.,* “Closed NL2SQL”), where the target database for generating and executing SQL is specified by the user.

Therefore, the Open NL2SQL problem introduces unique challenges that complicate the translation of natural language queries into SQL across multiple and heterogeneous databases. These challenges include (1) *database retrieval*, where relevant databases must be accurately identified and retrieved from a vast pool of data sources, (2) *handling heterogeneous schemas*, which requires integrating data with varying structures and terminologies, demanding sophisticated schema matching and linking techniques, (3) *answer aggregation*, involving the combination of several SQL queries from different databases, requiring robust methods to plan the query order, resolve conflicts, and ensure consistency, (4) *domain adaptation*, crucial for generalizing models across different domains and addressing variations in terminology and structure, (5) *scalability and efficiency*, as the system must handle large volumes of data while maintaining performance, and (6) *evaluating* *and benchmarking open* NL2SQL *solutions*, which requires comprehensive metrics and datasets that reflect real-world complexity.

B. Develop Cost-effective NL2SQL Methods

LLM-based NL2SQL methods show significant promise but are hindered by high token consumption, affecting both costs and inference times. On the other hand, our review shows that PLM-based NL2SQL methods excel at processing complex SQL queries and accurately interpreting database schemas.

Therefore, a promising direction is to combine the best of both worlds. In other words, we can develop modular NL2SQL solutions or utilize a multi-agent framework to interleave LLMs and PLMs for the NL2SQL task. This hybrid strategy may benefit the handling of complex queries while conserving tokens and reducing costs. However, effectively integrating and balancing the use of LLMs and PLMs to optimize both performance and resource efficiency is still an open problem.

C. Make NL2SQL Solutions Trustworthy

Developing trustworthy NL2SQL solutions is crucial for ensuring the accuracy and reliability of generated SQL queries, which can reduce risks and minimize the need for manual intervention. Next, we will discuss several research topics.

Interpreting NL2SQL Solutions. Understanding why a NL2SQL model performs a specific way can greatly enhance our confidence in its reliability. For example, we can employ Explainable AI techniques [149], [150] such as surrogate models [151] and saliency maps [152] to understand the decision-making process of the model. However, the effectiveness of applying these techniques in the NL2SQL setting is still an unknown question, especially with the combined use of LLMs and PLMs.

On the other hand, existing research explores employing multi-agent frameworks with LLMs to improve the reliability of NL2SQL solutions. A multi-agent framework can divide the NL2SQL task into specialized sub-tasks handled by different agents, each optimized for its specific role. This approach can enhance the overall robustness and accuracy of the system by leveraging the strengths of various agents. However, effectively coordinating multiple agents or models, ensuring they work harmoniously, and optimizing their collective performance remain challenging and open research questions.

NL2SQL Debugging Tools: Inspired by code compilers that provide step-by-step debugging functionality, developing a visual debugger for NL2SQL could significantly enhance the accuracy and reliability of NL2SQL systems. Such a tool would measure both the semantic and syntactic errors of the generated SQL queries. NL2SQL debugging tools would detect potentially erroneous SQL queries generated by the model and allow users to step through the SQL generation process, identify mismatches, and understand the logic behind the generated SQL. However, achieving this goal presents significant challenges. Traditional code compilers primarily capture syntactic errors, while NL2SQL debugging must also address semantic errors, *i.e.,* ensuring that the generated SQL query accurately reflects the intent of the NL query.

D. NL2SQL with Ambiguous and Unspecified NL Queries

Deploying NL2SQL solutions in real-world scenarios requires addressing the challenge of processing ambiguous and unspecified NL queries, as these are common in user interactions. We discuss several strategies to alleviate this issue.

NL *Query Rewriter* aims to refine NL queries for clarity and specificity. This process involves transforming vague or ambiguous user inputs into well-defined and precise queries. Future research should focus on developing methods to automatically or interactively refine NL queries with user input, leveraging database schema knowledge. By understanding the structure and constraints of the database, the rewriter can suggest modifications that align the NL with the underlying schema, ensuring that the queries input into the system are as unambiguous and specific as possible.

NL *Query Auto-completion* assists users in refining their queries by suggesting precise completions based on common patterns and historical data. This approach can help users formulate their queries more accurately by providing contextually relevant options. The challenge lies in developing an algorithm that effectively uses database context to produce clear, unambiguous NL queries. By analyzing the initial user input and incorporating information about the database schema and past query patterns, the auto-completion system can offer suggestions that guide users toward more precise and well-formed NL queries.

Training with Ambiguous and Unspecified NL *Queries.* If the model is trained or fine-tuned with a large amount of (NL, SQL) pairs, in which the NL queries are ambiguous and/or unspecified, undoubtedly, the NL2SQL solution would benefit greatly in terms of handling such cases. However, the cost of developing and curating such extensive datasets is high. One promising direction is to synthesize training data in an adaptive way, which will be introduced later.

E. Adaptive Training Data Synthesis

Current learning-based NL2SQL methods struggle to adapt to new, unseen domains, highlighting a gap in generalizability across data domains. In addition, the performance of these methods heavily relies on the quality, coverage, diversity, and amount of the training data.

Therefore, an interesting research problem is to automatically and incrementally generate (NL, SQL) pairs based on the model performance. The key idea is to dynamically synthesize or augment the training data by leveraging feedback from NL2SQL evaluation results. Specifically, by incorporating insights from evaluation metrics and evaluation results, we can identify specific weaknesses of the model. Using this information, we can synthesize training data that continually evolves with the help of LLMs to cover a broader range of scenarios and domains. This adaptive training data synthesis method may enhance the model's ability to generalize and perform accurately in diverse contexts, leading to more robust and versatile NL2SQL solutions.

XI. Conclusion

In this paper, we provide a comprehensive review of NL2SQL techniques from a lifecycle perspective in the era of LLMs. We formally state the NL2SQL task, discuss its challenges, and present a detailed taxonomy of solutions based on the language models they rely on. We summarize the key modules of language model-powered NL2SQL methods, covering pre-processing strategies, translation models, and post-processing techniques. We also analyze NL2SQL benchmarks and evaluation metrics, highlighting their characteristics and typical errors. Furthermore, we outline a roadmap for practitioners to adapt LLMs for NL2SQL solutions. Finally, we maintain an updated online handbook to guide researchers and practitioners in the latest NL2SQL advancements and discuss the research challenges and open problems for NL2SQL.

References

[1] 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," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–28, 2023.

[2] Z. Chen, S. Chen, M. White, R. J. Mooney, and et al., "Text-to-sql error correction with language models of code," in *ACL*, 2023.

[3] L. Wang and et al., "Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing," in *KDD*, 2022.

[4] A. Liu, X. Hu, L. Lin, and L. Wen, "Semantic enhanced text-to-sql parsing via iteratively learning schema linking graph," in *KDD*, 2022.

[5] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[6] D. Gao, H. Wang, Y. Li, and et al., "Text-to-sql empowered by large language models: A benchmark evaluation," *Proc. VLDB Endow.*, 2024.

[7] H. Li, J. Zhang, C. Li, and H. Chen, "Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql," 2023.

[8] N. Tang, C. Yang, J. Fan, L. Cao, Y. Luo, and A. Y. Halevy, "Verifai: Verified generative AI," in *CIDR*. www.cidrdb.org, 2024.

[9] Y. Ye, J. Hao, Y. Hou, Z. Wang, S. Xiao, Y. Luo, and W. Zeng, "Generative ai for visualization: State of the art and future directions," *Visual Informatics*, vol. 8, no. 2, pp. 43–66, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2468502X24000160

[10] Y. Zhu, S. Du, B. Li, Y. Luo, and N. Tang, "Are large language models good statisticians?" *CoRR*, vol. abs/2406.07815, 2024.

[11] Y. Xie, Y. Luo, G. Li, and N. Tang, "Haichart: Human and AI paired visualization system," *Proc. VLDB Endow.*, 2023.

[12] L. Shen, E. Shen, Y. Luo, X. Yang, X. Hu, X. Zhang, Z. Tai, and J. Wang, "Towards natural language interfaces for data visualization: A survey," *IEEE Trans. Vis. Comput. Graph.*, vol. 29, no. 6, pp. 3121–3144, 2023.

[13] Y. Luo, X. Qin, C. Chai, N. Tang, G. Li, and W. Li, "Steerable self-driving data visualization," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 475–490, 2022.

[14] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin, "Natural language to visualization by neural machine translation," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 1, pp. 217–226, 2022.

[15] J. Tang, Y. Luo, M. Ouzzani, G. Li, and H. Chen, "Sevi: Speech-to-visualization through neural machine translation," in *SIGMOD Conference*. ACM, 2022, pp. 2353–2356.

[16] Y. Luo, N. Tang, G. Li, C. Chai, W. Li, and X. Qin, "Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks," in *SIGMOD Conference*. ACM, 2021, pp. 1235–1247.

[17] X. Qin, Y. Luo, N. Tang, and G. Li, "Making data visualization more efficient and effective: a survey," *VLDB J.*, vol. 29, no. 1, pp. 93–117, 2020.

[18] Y. Luo, X. Qin, N. Tang, and G. Li, "Deepeye: Towards automatic data visualization," in *ICDE*. IEEE Computer Society, 2018, pp. 101–112.

[19] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang, "Deepeye: Creating good data visualizations by keyword search," in *SIGMOD Conference*. ACM, 2018, pp. 1733–1736.

[20] X. Zhou, Z. Sun, and G. Li, "DB-GPT: large language model meets database," *Data Sci. Eng.*, vol. 9, no. 1, pp. 102–111, 2024.

[21] S. Amer-Yahia, A. Bonifati, L. Chen, G. Li, K. Shim, J. Xu, and X. Yang, "From large language models to databases and back: A discussion on research and education," *SIGMOD Rec.*, vol. 52, no. 3, pp. 49–56, 2023.

[22] W. Zhang, Y. Wang, Y. Song, V. J. Wei, Y. Tian, Y. Qi, J. H. Chan, R. C.-W. Wong, and H. Yang, "Natural language interfaces for tabular data querying and visualization: A survey," *IEEE Transactions on Knowledge & Data Engineering*, no. 01, pp. 1–20, 2024.

[23] G. Katsogiannis-Meimarakis and G. Koutrika, "A survey on deep learning approaches for text-to-sql," *The VLDB Journal*, vol. 32, no. 4, pp. 905–936, 2023.

[24] N. Deng, Y. Chen, and Y. Zhang, "Recent advances in text-to-SQL: A survey of what we have and what we expect," in *Proceedings of the 29th International Conference on Computational Linguistics*, N. Calzolari, C.-R. Huang, H. Kim, J. Pustejovsky, L. Wanner, K.-S. Choi, P.-M. Ryu, H.-H. Chen, L. Donatelli, H. Ji, S. Kurohashi, P. Paggio, N. Xue, S. Kim, Y. Hahm, Z. He, T. K. Lee, E. Santus, F. Bond, and S.-H. Na, Eds. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 2166–2187. [Online]. Available: https://aclanthology.org/2022.coling-1.190

[25] H. Kim, B.-H. So, W.-S. Han, and H. Lee, "Natural language to sql: Where are we today?" *Proceedings of the VLDB Endowment*, vol. 13, no. 10, pp. 1737–1750, 2020.

[26] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang, "Next-generation database interfaces: A survey of llm-based text-to-sql," *arXiv preprint arXiv:2406.08426*, 2024.

[27] F. Őzcan, A. Quamar, J. Sen, and et al., "State of the art and open challenges in natural language interfaces to data," in *ACM SIGMOD*, 2020.

[28] Y. Li and D. Rafiei, "Natural language data management and interfaces: Recent development and open challenges," in *SIGMOD Conference*, 2017.

[29] G. Katsogiannis-Meimarakis and et al., "Natural language interfaces for databases with deep learning," *Proc. VLDB Endow.*, 2023.

[30] G. Katsogiannis-Meimarakis and G. Koutrika, "A deep dive into deep learning approaches for text-to-sql systems," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2846–2851.

[31] 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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Association for Computational Linguistics, 2018, pp. 3911–3921. [Online]. Available: https://doi.org/10.18653/v1/d18-1425

[32] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[33] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.06196

[34] N. Rajkumar and et al., "Evaluating the text-to-sql capabilities of large language models," *CoRR*, vol. abs/2204.00498, 2022.

[35] F. Li and H. V. Jagadish, "Nalir: an interactive natural language interface for querying relational databases," in *ACM SIGMOD*, ser. SIGMOD '14, 2014.

[36] T. Yu, C.-S. Wu, X. V. Lin, B. Wang, Y. C. Tan, X. Yang, D. Radev, R. Socher, and C. Xiong, "Grappa: Grammar-augmented pre-training for table semantic parsing," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://arxiv.org/abs/2009.13845

[37] C. Xiao, M. Dymetman, and C. Gardent, "Sequence-based structured prediction for semantic parsing," in *ACL*, 2016, pp. 1341–1350.

[38] K. Lin, B. Bogin, M. Neumann, J. Berant, and M. Gardner, "Grammar-based neural text-to-sql generation," *arXiv preprint arXiv:1905.13326*, 2019.

[39] B. Bogin, M. Gardner, and J. Berant, "Representing schema structure with graph neural networks for text-to-sql parsing," *arXiv preprint arXiv:1905.06241*, 2019.

[40] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *CoRR*, vol. abs/1709.00103, 2017. [Online]. Available: http://arxiv.org/abs/1709.00103

[41] X. Xu, C. Liu, and D. Song, “Sqlnet: Generating structured queries from natural language without reinforcement learning,” *arXiv preprint arXiv:1711.04436*, 2017.

[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[43] 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,” *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[44] J. Li and et al., “Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing,” *arXiv:2301.07507*, 2023.

[45] 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 abs/2306.08891 (2023),” *arXiv preprint arXiv:2306.08891*, 2023.

[46] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang, “The dawn of natural language to sql: Are we fully ready?” *Proc. VLDB Endow.*, 2024.

[47] H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, and H. Chen, “Codes: Towards building open-source language models for text-to-sql,” *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–28, 2024.

[48] C. Zhang, Y. Mao, Y. Fan, Y. Mi, Y. Gao, L. Chen, D. Lou, and J. Lin, “Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis,” in *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago AA, Chile, June 9-15, 2024*, P. Barceló, N. S. Pi, A. Meliou, and S. Sudarshan, Eds. ACM, 2024, pp. 93–105. [Online]. Available: https://doi.org/10.1145/3626246.3653375

[49] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, “Starcoder: may the source be with you!” *arXiv preprint arXiv:2305.06161*, 2023.

[50] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. Chang, F. Huang, R. Cheng, and Y. Li, “Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls,” in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

[51] W. Lei, W. Wang, Z. Ma, T. Gan, W. Lu, M.-Y. Kan, and T.-S. Chua, “Re-examining the role of schema linking in text-to-sql,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 6943–6954.

[52] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi, “Chess: Contextual harnessing for efficient sql synthesis,” *arXiv preprint arXiv:2405.16755*, 2024.

[53] M. Pourreza and D. Rafiei, “Dts-sql: Decomposed text-to-sql with small large language models,” *arXiv preprint arXiv:2402.01117*, 2024.

[54] G. Qu, J. Li, B. Li, B. Qin, N. Huo, C. Ma, and R. Cheng, “Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation,” *arXiv preprint arXiv:2405.15307*, 2024.

[55] Z. Li, X. Wang, J. Zhao, S. Yang, G. Du, X. Hu, B. Zhang, Y. Ye, Z. Li, R. Zhao *et al.*, “Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency,” *arXiv preprint arXiv:2403.09732*, 2024.

[56] H. Zhang, R. Cao, H. Xu, L. Chen, and K. Yu, “Coe-sql: In-context learning for multi-turn text-to-sql with chain-of-editions,” *arXiv preprint arXiv:2405.02712*, 2024.

[57] T. Ren, Y. Fan, Z. He, R. Huang, J. Dai, C. Huang, Y. Jing, K. Zhang, Y. Yang, and X. S. Wang, “Purple: Making a large language model a better sql writer,” 2024. [Online]. Available: https://arxiv.org/abs/2403.20014

[58] 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,” 2024. [Online]. Available: https://arxiv.org/abs/2402.17144

[59] Y. Xie, X. Jin, T. Xie, M. Lin, L. Chen, C. Yu, L. Cheng, C. Zhuo, B. Hu, and Z. Li, “Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm,” *arXiv preprint arXiv:2402.10671*, 2024.

[60] X. Dong, C. Zhang, Y. Ge, and et al., “C3: Zero-shot text-to-sql with chatgpt,” *arXiv preprint arXiv:2307.07306*, 2023.

[61] D. Rai, B. Wang, Y. Zhou, and Z. Yao, “Improving generalization in language model-based text-to-SQL semantic parsing: Two simple semantic boundary-based techniques,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 150–160. [Online]. Available: https://aclanthology.org/2023.acl-short.15

[62] H. Zhang, R. Cao, L. Chen, H. Xu, and K. Yu, “Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 3501–3532.

[63] S. Chang and E. Fosler-Lussier, “Selective demonstrations for cross-domain text-to-sql,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 14 174–14 189.

[64] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, Q.-W. Zhang, Z. Yan, and Z. Li, “Mac-sql: A multi-agent collaborative framework for text-to-sql.” *CoRR*, 2023.

[65] H. Fu, C. Liu, B. Wu, F. Li, J. Tan, and J. Sun, “Catsql: Towards real world natural language to sql applications,” *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1534–1547, 2023.

[66] A. Bazaga, P. Liò, and G. Micklem, “Sqlformer: Deep auto-regressive query graph generation for text-to-sql translation,” *arXiv preprint arXiv:2310.18376*, 2023.

[67] Y. Xiang, Q.-W. Zhang, X. Zhang, Z. Liu, Y. Cao, and D. Zhou, “G3r: A graph-guided generate-and-rerank framework for complex and cross-domain text-to-sql generation,” in *Findings of the Association for Computational Linguistics: ACL 2023*, 2023, pp. 338–352.

[68] Y. Hu, Y. Zhao, J. Jiang, W. Lan, H. Zhu, A. Chauhan, A. H. Li, L. Pan, J. Wang, C. Hang, S. Zhang, J. Guo, M. Dong, J. Lilien, P. Ng, Z. Wang, V. Castelli, and B. Xiang, “Importance of synthesizing high-quality data for text-to-sql parsing,” in *ACL (Findings)*. Association for Computational Linguistics, 2023, pp. 1327–1343.

[69] L. Zeng, S. H. K. Parthasarathi, and D. Hakkani-Tur, “N-best hypotheses reranking for text-to-sql systems,” in *2022 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2023, pp. 663–670.

[70] 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,” *arXiv preprint arXiv:2205.06983*, 2022.

[71] T. Scholak and et al., “Picard: Parsing incrementally for constrained auto-regressive decoding from language models,” 2021.

[72] C. Gao, B. Li, W. Zhang, W. Lam, B. Li, F. Huang, L. Si, and Y. Li, “Towards generalizable and robust text-to-sql parsing,” *arXiv preprint arXiv:2210.12674*, 2022.

[73] B. Hui, R. Geng, L. Wang, B. Qin, Y. Li, B. Li, J. Sun, and Y. Li, “S^{2}SQL: Injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers,” in *Findings of the Association for Computational Linguistics: ACL 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1254–1262. [Online]. Available: https://aclanthology.org/2022.findings-acl.99

[74] 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*. Association for Computational Linguistics, 2020, pp. 7567–7578.

[75] O. Rubin and J. Berant, “Smbop: Semi-autoregressive bottom-up semantic parsing,” *arXiv preprint arXiv:2010.12412*, 2020.

[76] J. Huang, Y. Wang, Y. Wang, Y. Dong, and Y. Xiao, “Relation aware semi-autoregressive semantic parsing for nl2sql,” *arXiv preprint arXiv:2108.00804*, 2021.

[77] X. V. Lin, R. Socher, and C. Xiong, “Bridging textual and tabular data for cross-domain text-to-sql semantic parsing,” *arXiv preprint arXiv:2012.12627*, 2020.

[78] 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,” *CoRR*, vol. abs/1905.08205, 2019. [Online]. Available: http://arxiv.org/abs/1905.08205

[79] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. Radev, “Typesql: Knowledge-based type-aware neural text-to-sql generation,” *arXiv preprint arXiv:1804.09769*, 2018.

[80] F. J. Damerau, “A technique for computer detection and correction of spelling errors,” *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[81] U. Brunner and K. Stockinger, "Valuenet: A natural language-to-sql system that learns from database information," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 2177–2182.

[82] Z. Dong, S. Sun, H. Liu, J.-G. Lou, and D. Zhang, "Data-anonymous encoding for text-to-sql generation," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 5405–5414.

[83] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[84] D. Lee, C. Park, J. Kim, and H. Park, "Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation," *arXiv preprint arXiv:2405.07467*, 2024.

[85] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. Chi, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *CoRR*, vol. abs/2201.11903, 2022. [Online]. Available: https://arxiv.org/abs/2201.11903

[86] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[87] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," *arXiv preprint arXiv:2005.08314*, 2020.

[88] R. Speer, C. Havasi *et al.*, "Representing general relational knowledge in conceptnet 5." in *LREC*, vol. 2012, 2012, pp. 3679–86.

[89] 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," *arXiv preprint arXiv:2106.01093*, 2021.

[90] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 604–613.

[91] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, no. 4, p. 333–389, apr 2009. [Online]. Available: https://doi.org/10.1561/1500000019

[92] L. Dou, Y. Gao, X. Liu, M. Pan, D. Wang, W. Che, D. Zhan, M.-Y. Kan, and J.-G. Lou, "Towards knowledge-intensive text-to-sql semantic parsing with formulaic knowledge," *arXiv preprint arXiv:2301.01067*, 2023.

[93] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," *arXiv preprint arXiv:2004.04906*, 2020.

[94] G. Sui, Z. Li, Z. Li, S. Yang, J. Ruan, H. Mao, and R. Zhao, "Reboost large language model-based text-to-sql, text-to-python, and text-to-function–with real applications in traffic domain," *arXiv preprint arXiv:2310.18752*, 2023.

[95] F. Li, T. Pan, and H. V. Jagadish, "Schema-free sql," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 1051–1062.

[96] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li, and D. Radev, "Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task," *arXiv preprint arXiv:1810.05237*, 2018.

[97] J.-O. Lee and D.-K. Baik, "Semql: a semantic query language for multidatabase systems," in *Proceedings of the eighth international conference on Information and knowledge management*, 1999, pp. 259–266.

[98] R. Zhang, T. Yu, H. Y. Er, S. Shim, E. Xue, X. V. Lin, T. Shi, C. Xiong, R. Socher, and D. Radev, "Editing-based sql query generation for cross-domain context-dependent questions," *arXiv preprint arXiv:1909.00786*, 2019.

[99] Y. Gan, X. Chen, J. Xie, M. Purver, J. R. Woodward, J. Drake, and Q. Zhang, "Natural SQL: Making SQL easier to infer from natural language specifications," in *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2030–2042. [Online]. Available: https://aclanthology.org/2021.findings-emnlp.174

[100] T. Wolfson, M. Geva, A. Gupta, M. Gardner, Y. Goldberg, D. Deutch, and J. Berant, "Break it down: A question understanding benchmark," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 183–198, 2020.

[101] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[102] X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu *et al.*, "Deepseek llm: Scaling open-source language models with longtermism," *arXiv preprint arXiv:2401.02954*, 2024.

[103] S. Qiao, Y. Ou, N. Zhang, X. Chen, Y. Yao, S. Deng, C. Tan, F. Huang, and H. Chen, "Reasoning with language model prompting: A survey," *arXiv preprint arXiv:2212.09597*, 2022.

[104] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[105] X. Liu, J. Wang, J. Sun, X. Yuan, G. Dong, P. Di, W. Wang, and D. Wang, "Prompting frameworks for large language models: A survey," *arXiv preprint arXiv:2311.12785*, 2023.

[106] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[107] C.-Y. Tai, Z. Chen, T. Zhang, X. Deng, and H. Sun, "Exploring chain-of-thought style prompting for text-to-sql," *arXiv preprint arXiv:2305.14215*, 2023.

[108] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proc. VLDB Endow.*, vol. 8, no. 1, pp. 73–84, 2014. [Online]. Available: http://www.vldb.org/pvldb/vol8/p73-li.pdf

[109] B. Eyal, A. Bachar, O. Haroche, M. Mahabi, and M. Elhadad, "Semantic decomposition of question and sql for text-to-sql parsing," *arXiv preprint arXiv:2310.13575*, 2023.

[110] T. Wolfson, J. Berant, and D. Deutch, "Weakly supervised mapping of natural language to sql through question decomposition," *Clin. Orthop. Relat. Res*, 2021.

[111] D. Lee, "Clause-wise and recursive decoding for complex and cross-domain text-to-sql generation," *arXiv preprint arXiv:1904.08835*, 2019.

[112] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[113] M. Renze and E. Guven, "The effect of sampling temperature on problem solving in large language models," *arXiv preprint arXiv:2402.05201*, 2024.

[114] A. Kelkar, R. Relan, V. Bhardwaj, S. Vaichal, C. Khatri, and P. Relan, "Bertrand-dr: Improving text-to-sql using a discriminative re-ranker," *arXiv preprint arXiv:2002.00557*, 2020.

[115] Y. Luo, Y. Zhou, N. Tang, G. Li, C. Chai, and L. Shen, "Learned data-aware image representations of line charts for similarity search," *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 88:1–88:29, 2023.

[116] K. Zhang, X. Lin, Y. Wang, X. Zhang, F. Sun, C. Jianhe, H. Tan, X. Jiang, and H. Shen, "Refsql: A retrieval-augmentation framework for text-to-sql generation," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 664–673.

[117] D. A. Dahl, M. Bates, M. Brown, W. M. Fisher, K. Hunicke-Smith, D. S. Pallett, C. Pao, A. I. Rudnicky, and E. Shriberg, "Expanding the scope of the ATIS task: The ATIS-3 corpus," in *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*. Morgan Kaufmann, 1994. [Online]. Available: https://aclanthology.org/H94-1010/

[118] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, W. J. Clancey and D. S. Weld, Eds. AAAI Press / The MIT Press, 1996, pp. 1050–1055. [Online]. Available: http://www.aaai.org/Library/AAAI/1996/aaai96-156.php

[119] L. R. Tang and R. J. Mooney, "Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing," in *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, EMNLP 2000, Hong Kong, October 7-8, 2000*, H. Schütze and K. Su, Eds. Association for Computational Linguistics, 2000, pp. 133–141. [Online]. Available: https://aclanthology.org/W00-1317/

[120] N. Yaghmazadeh, Y. Wang, I. Dillig, and T. Dillig, "Sqlizer: query synthesis from natural language," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 63:1–63:26, 2017. [Online]. Available: https://doi.org/10.1145/3133887

[121] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, "Learning a neural semantic parser from user feedback," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, R. Barzilay and M. Kan,

Eds. Association for Computational Linguistics, 2017, pp. 963–973. [Online]. Available: https://doi.org/10.18653/v1/P17-1089

[122] C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. R. Radev, "Improving text-to-sql evaluation methodology," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, I. Gurevych and Y. Miyao, Eds. Association for Computational Linguistics, 2018, pp. 351–360. [Online]. Available: https://aclanthology.org/P18-1033/

[123] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, J. Kraft, V. Zhang, C. Xiong, R. Socher, and D. R. Radev, "Sparc: Cross-domain semantic parsing in context," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, A. Korhonen, D. R. Traum, and L. Màrquez, Eds. Association for Computational Linguistics, 2019, pp. 4511–4523. [Online]. Available: https://doi.org/10.18653/v1/p19-1443

[124] T. Yu, R. Zhang, H. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, Y. Jiang, M. Yasunaga, S. Shim, T. Chen, A. R. Fabbri, Z. Li, L. Chen, Y. Zhang, S. Dixit, V. Zhang, C. Xiong, R. Socher, W. S. Lasecki, and D. R. Radev, "Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 1962–1979. [Online]. Available: https://doi.org/10.18653/v1/D19-1204

[125] Q. Min, Y. Shi, and Y. Zhang, "A pilot study for chinese SQL semantic parsing," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 3650–3656. [Online]. Available: https://doi.org/10.18653/v1/D19-1377

[126] P. Wang, T. Shi, and C. K. Reddy, "Text-to-sql generation for question answering on electronic medical records," in *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Y. Huang, I. King, T. Liu, and M. van Steen, Eds. ACM / IW3C2, 2020, pp. 350–361. [Online]. Available: https://doi.org/10.1145/3366423.3380120

[127] T. Shi, C. Zhao, J. L. Boyd-Graber, H. D. III, and L. Lee, "On the potential of lexico-logical alignments for semantic parsing to SQL queries," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1849–1864. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.167

[128] J. Sen, C. Lei, A. Quamar, F. Özcan, V. Efthymiou, A. Dalmia, G. Stager, A. Mittal, D. Saha, and K. Sankaranarayanan, "Athena++ natural language querying for complex nested sql queries," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2747–2759, 2020.

[129] A. T. Nguyen, M. H. Dao, and D. Q. Nguyen, "A pilot study of text-to-sql semantic parsing for vietnamese," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 4079–4085. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.364

[130] L. Wang, A. Zhang, K. Wu, K. Sun, Z. Li, H. Wu, M. Zhang, and H. Wang, "Dusql: A large-scale and pragmatic chinese text-to-sql dataset," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds. Association for Computational Linguistics, 2020, pp. 6923–6935. [Online]. Available: https://doi.org/10.18653/v1/2020.emnlp-main.562

[131] M. A. José and F. G. Cozman, "mrat-sql+gap: A portuguese text-to-sql transformer," in *Intelligent Systems - 10th Brazilian Conference, BRACIS 2021, Virtual Event, November 29 - December 3, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, A. Britto and K. V. Delgado, Eds., vol. 13074. Springer, 2021, pp. 511–525. [Online]. Available: https://doi.org/10.1007/978-3-030-91699-2_35

[132] J. Guo, Z. Si, Y. Wang, Q. Liu, M. Fan, J. Lou, Z. Yang, and T. Liu, "Chase: A large-scale and pragmatic chinese dataset for cross-database context-dependent text-to-sql," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, 2021, pp. 2316–2331. [Online]. Available: https://doi.org/10.18653/v1/2021.acl-long.180

[133] 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 *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, 2021, pp. 2505–2515. [Online]. Available: https://doi.org/10.18653/v1/2021.acl-long.195

[134] Y. Gan, X. Chen, and M. Purver, "Exploring underexplored limitations of cross-domain text-to-sql generalization," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds. Association for Computational Linguistics, 2021, pp. 8926–8931. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.702

[135] X. Deng, A. H. Awadallah, C. Meek, O. Polozov, H. Sun, and M. Richardson, "Structure-grounded pretraining for text-to-sql," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tür, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, Eds. Association for Computational Linguistics, 2021, pp. 1337–1350. [Online]. Available: https://doi.org/10.18653/v1/2021.naacl-main.105

[136] C. Lee, O. Polozov, and M. Richardson, "Kaggledbqa: Realistic evaluation of text-to-sql parsers," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, 2021, pp. 2261–2273. [Online]. Available: https://doi.org/10.18653/v1/2021.acl-long.176

[137] M. Hazoom, V. Malik, and B. Bogin, "Text-to-sql in the wild: A naturally-occurring dataset based on stack exchange data," *CoRR*, vol. abs/2106.05006, 2021. [Online]. Available: https://arxiv.org/abs/2106.05006

[138] P. Ma and S. Wang, "Mt-teql: Evaluating and augmenting neural NLIDB on real-world linguistic and schema variations," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 569–582, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol15/p569-ma.pdf

[139] D. Bakshandaeva, O. Somov, E. Dmitrieva, V. Davydova, and E. Tutubalina, "PAUQ: text-to-sql in russian," in *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Association for Computational Linguistics, 2022, pp. 2355–2376. [Online]. Available: https://doi.org/10.18653/v1/2022.findings-emnlp.175

[140] L. Dou, Y. Gao, X. Liu, M. Pan, D. Wang, W. Che, D. Zhan, M.-Y. Kan, and J.-G. Lou, "Towards knowledge-intensive text-to-SQL semantic parsing with formulaic knowledge," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5240–5253. [Online]. Available: https://aclanthology.org/2022.emnlp-main.350

[141] S. Chang, J. Wang, M. Dong, L. Pan, H. Zhu, A. H. Li, W. Lan, S. Zhang, J. Jiang, J. Lilien, S. Ash, W. Y. Wang, Z. Wang, V. Castelli, P. Ng, and B. Xiang, "Dr.spider: A diagnostic evaluation benchmark towards text-to-sql robustness," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=Wc5bmZZU9cy

[142] A. Bhaskar, T. Tomar, A. Sathe, and S. Sarawagi, "Benchmarking and improving text-to-sql generation under ambiguity," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Association for Computational Linguistics, 2023, pp. 7053–7074. [Online]. Available: https://doi.org/10.18653/v1/2023.emnlp-main.436

[143] Y. Zhang, J. Deriu, G. Katsogiannis-Meimarakis, C. Kosten, G. Koutrika, and K. Stockinger, "Sciencebenchmark: A complex real-

world benchmark for evaluating natural language to SQL systems," *Proc. VLDB Endow.*, vol. 17, no. 4, pp. 685–698, 2023. [Online]. Available: https://www.vldb.org/pvldb/vol17/p685-stockinger.pdf

[144] R. Kumar, A. R. Dibbu, S. Harsola, V. Subrahmaniam, and A. Modi, "Booksql: A large scale text-to-sql dataset for accounting domain," 2024. [Online]. Available: https://arxiv.org/abs/2406.07860

[145] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *CoRR*, vol. abs/1709.00103, 2017. [Online]. Available: http://arxiv.org/abs/1709.00103

[146] A. Narechania, A. Fourney, B. Lee, and G. Ramos, "Diy: Assessing the correctness of natural language to sql systems," in *26th International Conference on Intelligent User Interfaces*, 2021, pp. 597–607.

[147] Z. Ning, Z. Zhang, T. Sun, Y. Tian, T. Zhang, and T. J.-J. Li, "An empiricmacsqlal study of model errors and user error discovery and repair strategies in natural language database queries," in *Proceedings of the 28th International Conference on Intelligent User Interfaces*, 2023, pp. 633–649.

[148] R. Sun, S. O. Arik, H. Nakhost, H. Dai, R. Sinha, P. Yin, and T. Pfister, "Sql-palm: Improved large language modeladaptation for text-to-sql," *arXiv preprint arXiv:2306.00739*, 2023.

[149] S. Ali, T. Abuhmed, S. H. A. El-Sappagh, K. Muhammad, J. M. Alonso-Moral, R. Confalonieri, R. Guidotti, J. D. Ser, N. D. Rodríguez, and F. Herrera, "Explainable artificial intelligence (XAI): what we know and what is left to attain trustworthy artificial intelligence," *Inf. Fusion*, vol. 99, p. 101805, 2023.

[150] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, and M. Du, "Explainability for large language models: A survey," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 2, pp. 20:1–20:38, 2024.

[151] Y. Chen, R. Zhong, N. Ri, C. Zhao, H. He, J. Steinhardt, Z. Yu, and K. R. McKeown, "Do models explain themselves? counterfactual simulatability of natural language explanations," *CoRR*, vol. abs/2307.08678, 2023.

[152] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 3145–3153.