Vicinagearth

RESEARCH Open Access

Rethinking data in NL2SQL: a survey of what we have and what we expect

Yuankai Fan[1*], Qizhen Weng[1], Yin Chen[1] and X. Sean Wang[2]

Abstract
Natural Language to SQL (NL2SQL) has become a cornerstone task for enabling natural language interfaces to relational databases. With the emergence of large language models, NL2SQL systems have achieved remarkable performance gains. However, despite the focus on architectural innovations and benchmark achievements, we argue that NL2SQL is fundamentally a data-centric task — where the quality, structure, and utilization of data play a more critical role than often acknowledged. In this survey, we re-examine the NL2SQL landscape through the lens of how data are used throughout the system pipeline. Specifically, we offer a brief overview of the task challenges and evolutionary process of NL2SQL. Next, we categorize the major data types and analyze how these data sources are leveraged throughout the NL2SQL lifecycle. We then introduce the datasets and metrics used to evaluate NL2SQL systems. Finally, we highlight the remaining challenges and outline promising directions for future research. We hope that this survey can serve as a quick reference to existing work and motivate future research from a data perspective.

Keywords NL2SQL, Database interface, Large language models

1 Introduction

In the era of big data, a substantial portion of information is stored in relational databases, which form the backbone of data management systems across many organizations. As data volumes continue to grow, the ability to efficiently query and utilize this data has become a critical factor in maintaining a competitive edge across various industries. However, querying relational databases requires proficiency in SQL, a specialized skill that poses a barrier for non-technical users seeking to access and interact with the data.

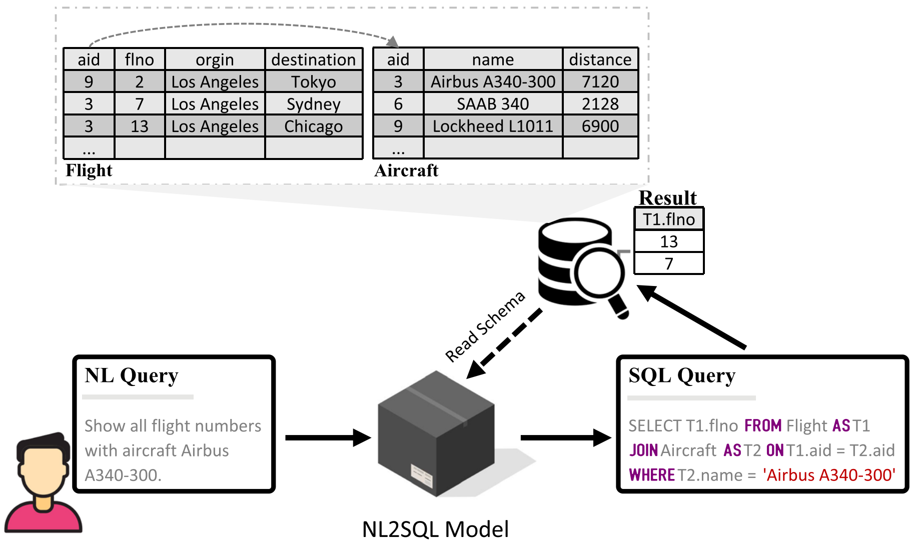
Natural language to SQL (i.e., NL2SQL, also known as Text-to-SQL), which converts a natural language (NL) query into a corresponding SQL query, can significantly lower the barrier for both lay users (and also expert users) in accessing massive datasets and deriving insights [1–13]. To illustrate, Fig. 1 presents the general workflow: An end-user who wants to query data about the flight numbers of a specific aircraft from a database containing two tables, named Flight and Aircraft. The user simply inputs the NL query "*Show all flight numbers with aircraft Airbus A340-330*", the NL2SQL model then automatically generates the corresponding SQL query for the user:

```
SELECT T1.flno FROM Flight AS T1
JOIN Aircraft AS T2 ON T1.aid = T2.aid
WHERE T2.name = 'Airbus A340-300'
```

After that, the SQL query can be executed against the database to get the results, namely flight numbers 7 and 13 in this example.

Researchers have made substantial progress in this area. Initially, template-based and rule-based methods [14–16] were employed. These approaches involved creating SQL templates for various scenarios. While

*Correspondence:
Yuankai Fan
fanyk1@chinatelecom.cn
[1] Institute of Artificial Intelligence (TeleAI), China Telecom, 199 Longwen Road, Xuhui District 200232, Shanghai, People's Republic of China
[2] College of Computer Science and Artificial Intelligence, Fudan University, Jiangwan Campus, Yangpu District 200438, Shanghai, People's Republic of China



aid	flno	orgin	destination
9	2	Los Angeles	Tokyo
3	7	Los Angeles	Sydney
3	13	Los Angeles	Chicago
...			

aid	name	distance
3	Airbus A340-300	7120
6	SAAB 340	2128
9	Lockheed L1011	6900
...		

T1.flno
13
7

```
SELECT T1.flno FROM Flight AS T1
JOIN Aircraft AS T2 ON T1.aid = T2.aid
WHERE T2.name = 'Airbus A340-300'
```

Fig. 1 Example of NL2SQL system workflow. Given the database schema and the user's utterance, the system outputs a corresponding SQL query to retrieve the result from the database

template-based methods showed promise, they required significant manual effort. With the rapid advancement of deep learning, Sequence-to-sequence (Seq2seq) methods [17–22] have emerged as the mainstream approach. Seq2Seq models provide an end-to-end solution that directly maps NL input to SQL output using an encoder-decoder architecture. Among Seq2Seq methods, pre-trained language models (PLMs), which serve as predecessors to large language models (LLMs), show promise in NL2SQL tasks. As model sizes and training data continue to grow, PLMs naturally evolve into LLMs, exhibiting even greater power. The remarkable capabilities of LLMs have prompted research into their application for NL2SQL tasks [5, 6, 8, 13, 23–27].

However, amid this prevailing model-centric momentum, a critical factor has received comparatively less attention: data. Unlike many natural language processing (NLP) tasks, where the input and output spaces are relatively fixed, NL2SQL operates at the intersection of unstructured NL with structured database schemas. This intersection makes the quality, diversity, and contextualization of data not just important but central to the success of such systems. In fact, the performance and generalizability of NL2SQL models are often constrained more by data limitations than by model architecture itself.

This survey aims to shift the perspective from model to data, rethinking NL2SQL through a data-centric lens. We argue that current bottlenecks in generalization, robustness, and real-world applicability often stem less from modeling limitations than from issues in how data are constructed, represented, and used. To this end, we systematically review recent NL2SQL techniques and present a comprehensive survey of how data is involved in the following aspects:

1. **Overview**. We provide a brief introduction to the technical challenges and evolutionary process of NL2SQL, the trajectory of NL2SQL, and a review of existing solutions, categorizing them into four major groups presented in Fig. 2.
2. **Data with NL2SQL**. We begin by categorizing the primary data types involved, then analyze how these data sources are leveraged in existing NL2SQL solutions.
3. **Benchmarks**. We describe the commonly used datasets and benchmarks for evaluating NL2SQL systems. We discuss their characteristics, complexity, and the challenges they pose for development and evaluation, along with the evaluation metrics.
4. **Analysis**. We conduct a comparative analysis of experimental results from existing studies to further investigate the practical value and applicability of methods and models.
5. **Summarization**: We summarize the limitations of existing NL2SQL solutions, with a focus on data utilization: what we have and what we expect. We outline future directions and opportunities for improvement.

To summarize our technical contributions, we present the first NL2SQL survey in the community from a data perspective, establishing a taxonomy of data types and usage patterns, highlighting their influence on model

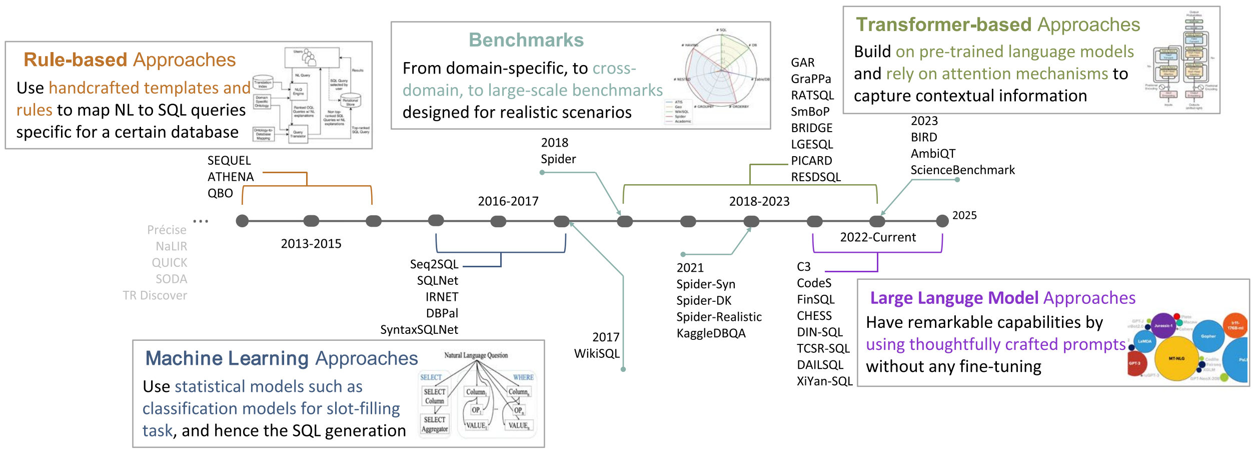

Rule-based Approaches
Use handcrafted templates and rules to map NL to SQL queries specific for a certain database
Benchmarks
From domain-specific, to cross-domain, to large-scale benchmarks designed for realistic scenarios
Transformer-based Approaches
Build on pre-trained language models and rely on attention mechanisms to capture contextual information
Machine Learning Approaches
Use statistical models such as classification models for slot-filling task, and hence the SQL generation
Large Languge Model Approaches
Have remarkable capabilities by using thoughtfully crafted prompts without any fine-tuning
Précise
NaLIR
QUICK
SODA
TR Discover
SEQUEL
ATHENA
QBO
2013-2015
2016-2017
Seq2SQL
SQLNet
IRNET
DBPal
SyntaxSQLNet
2017
WikiSQL
2018
Spider
2021
Spider-Syn
Spider-DK
Spider-Realistic
KaggleDBQA
2018-2023
GAR
GraPPa
RATSQL
SmBoP
BRIDGE
LGESQL
PICARD
RESDSQL
2022-Current
C3
CodeS
FinSQL
CHESS
DIN-SQL
TCSR-SQL
DAILSQL
XiYan-SQL
2023
BIRD
AmbiQT
ScienceBenchmark
2025

Fig. 2 Evolution of NL2SQL

design. Then, we analyze the task challenges and provide a more detailed and comprehensive summary of existing NL2SQL solutions and benchmarks. Finally, we highlight the remaining challenges in robustness and real-world deployment and offer actionable directions for future research.

2 Overview

In this section, we first formalize the definition of the NL2SQL task (Sect. 2.1). We then discuss the key challenges in this task (Sect. 2.2). Finally, we describe the evolution of NL2SQL systems and review existing systems accordingly (Sect. 2.3).

2.1 Task formulation

NL2SQL Translation. Formally, given a user question Q expressed in NL and a database D defined by a schema S and populated with instances I, the NL2SQL task aims to translate Q into a corresponding SQL query L using a translation model M:

$$L = M(Q, D) \tag{1}$$

The resulting query L should be executable on D to produce a result R that answers the original question q.

2.2 Task challenges

We examine the primary technical challenges involved in developing robust NL2SQL solutions for real-world applications:

C1: Ambiguous and Under-specified NL Query. NL queries often exhibit uncertainty due to ambiguity and under-specification [28]. In the context of the NL2SQL task, these problems can be categorized as follows:

- *Lexical Ambiguity*: Arises when a word has multiple meanings. For example, "apple" could refer to a fruit or a technology company.
- *Syntactic Ambiguity*: Occurs when a sentence permits multiple grammatical interpretations. For instance, in "I shot an elephant in my pajamas," it is unclear whether the speaker was wearing pajamas or the elephant was.
- *Under-specification*: Happens when expressions lack sufficient detail to determine a specific meaning, especially in everyday conversation. For example, if someone says "Let's meet at the station," it's unclear which station they mean without additional context.

C2: Database Complexity. The NL2SQL task demands a deep understanding of the database schema, including table names, columns, relationships, and data values. The complexity of modern schemas and the scale of data further compound the challenge.

- *Table Joins*: Queries often require retrieving information spread across multiple tables. This necessitates understanding how tables are related—typically through foreign keys—and determining the correct join conditions to produce meaningful results.
- *Ambiguity in Attributes and Values*: Ambiguous values and attributes in a database can make it difficult for systems to accurately determine the intended context.
- *Large Data Volumes*: Efficiently handling vast data volumes in large databases is critical, as processing all data as input is impractical [29]. Additionally, dirty data, such as missing values, duplicates, or inconsistencies, can lead to erroneous query results (e.g., affecting `WHERE` clauses) if not properly managed.

C3: NL2SQL Translation. The NL2SQL task differs fundamentally 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, DB and output SQL. Specifically, the NL2SQL task faces several unique challenges:

- *Unstructured NL vs. Structured SQL*: NL is inherently unstructured and often ambiguous, while SQL is a formal language with strict syntactic rules. Bridging this gap requires accurate interpretation of user intent and precise generation of executable queries.
- *One-to-Many Mapping*: A single NL query can correspond to multiple SQL queries that fulfill the query intent, leading to ambiguity in determining appropriate SQL translation.
- *NL2SQL Trustworthiness and Explainability*: Beyond accuracy, the system must produce queries that users can trust and understand. Explainability is crucial, as it allows users to verify how the NL input was interpreted and how the SQL query was constructed [9]. Transparent and interpretable translations help build confidence in the system's outputs and facilitate error detection and correction.

2.3 Evolutionary process

Over the years, the NL2SQL research field has made substantial progress in both the database and the NLP communities. It has transitioned from early rule-based solutions to deep learning techniques and, more recently, to the adoption of pretrained and large language models. Figure 2 provides a visual summary of this evolution:

1. **Rule-based Methods**: Early solutions relied heavily on rule-based methods [14–16, 30, 31], where manually crafted rules and heuristics were used to map NL questions to SQL queries. Rule-based systems excel at generating syntactically correct queries by relying on predefined templates and strict grammatical rules, ensuring adherence to SQL syntax and structure. Their deterministic nature guarantees consistent outputs for specific input patterns, minimizing variability in query generation. However, these methods struggle to interpret linguistically complex or ambiguous natural language questions, such as those involving nested clauses, coreference, or ellipsis, and often fail to map these elements to database schema components, such as tables or columns. Their reliance on manual rules limits generalization to unseen question patterns or schema structures, requiring frequent updates for new domains. Additionally, encoding complex schemas with multi-table relationships into usable rules is labor-intensive and error-prone.
2. **Machine Learning Methods**: With the rise of deep neural networks, sequence-to-sequence (Seq2Seq) models and encoder-decoder architectures were adopted to generate SQL queries from natural language input [17, 98]. SQLNet [18], for example, employs a sketch-based approach and frames the NL2SQL task as a slot-filling problem. TypeSQL [32] builds on SQLNet by incorporating type information extracted from either knowledge graphs or table content, enabling a better understanding of entities and numerical values. Bogin et al. [33] propose using graph neural networks (GNNs) to encode the database schema, thereby enriching the representation. IRNet [20] introduces an intermediate representation that captures higher-level abstractions than raw SQL and leverages custom type vectors to enhance the modeling of both natural language queries and database schemas. Despite their adaptability, these approaches are all based on the Seq2Seq generation paradigm, which struggles to effectively handle complex queries.
3. **PLM-based Methods**: PLMs have emerged as a powerful solution for NL2SQL, leveraging the vast amounts of linguistic knowledge and semantic understanding captured during the pre-training process. The early adoption of PLMs in NL2SQL primarily focused on fine-tuning off-the-shelf PLMs, such as BERT [34] and RoBERTa, on standard NL2SQL datasets. These PLMs, pre-trained on large amounts of training corpus, captured rich semantic representations and language understanding capabilities. By fine-tuning them on NL2SQL tasks, researchers aimed to leverage the semantic and linguistic understanding of PLMs to generate accurate SQL queries [19, 21, 35, 36]. Another line of research focuses on incorporating schema information into PLMs to improve their understanding of database structures and enable them to generate more executable SQL queries. Schema-aware PLMs are designed to capture the relationships and constraints present in the database structure [19]. While PLMs generate more accurate SQL than deep learning-based methods, they still struggle with complex operations such as outer joins and aggregations, often producing syntactically flawed queries. Besides that, cross-domain performance drops significantly when the schema or vocabulary differs from the training data, necessitating resource-intensive fine-tuning.
4. **LLM-based Methods**: LLMs, such as the GPT series [37, 38], have gained significant attention in recent years due to their ability to generate coherent and

fluent text. Researchers have begun exploring the potential of LLMs for NL2SQL by leveraging their extensive knowledge reserves and superior generation capabilities [6, 11–13, 25, 26, 39, 40]. These approaches often involve prompt engineering to guide proprietary LLMs in SQL generation [26, 39] or fine-tuning open-source LLMs on NL2SQL datasets [13]. The integration of LLMs in NL2SQL is still an emerging research area with significant potential for further exploration and improvement. Researchers are investigating ways to better leverage LLMs' knowledge and reasoning capabilities, incorporate domain-specific knowledge [8], and develop more efficient fine-tuning strategies [40]. As the field continues to evolve, we anticipate the development of more advanced and superior LLM-based implementations that will elevate the performance and generalization of NL2SQL to new heights.

3 Data with NL2SQL

Data plays a central role in the NL2SQL task, as the quality and structure of the data directly impact the system's performance and accuracy [41–43]. In this section, we first introduce the five key data types involved and then examine how each type contributes to different stages of the NL2SQL pipeline.

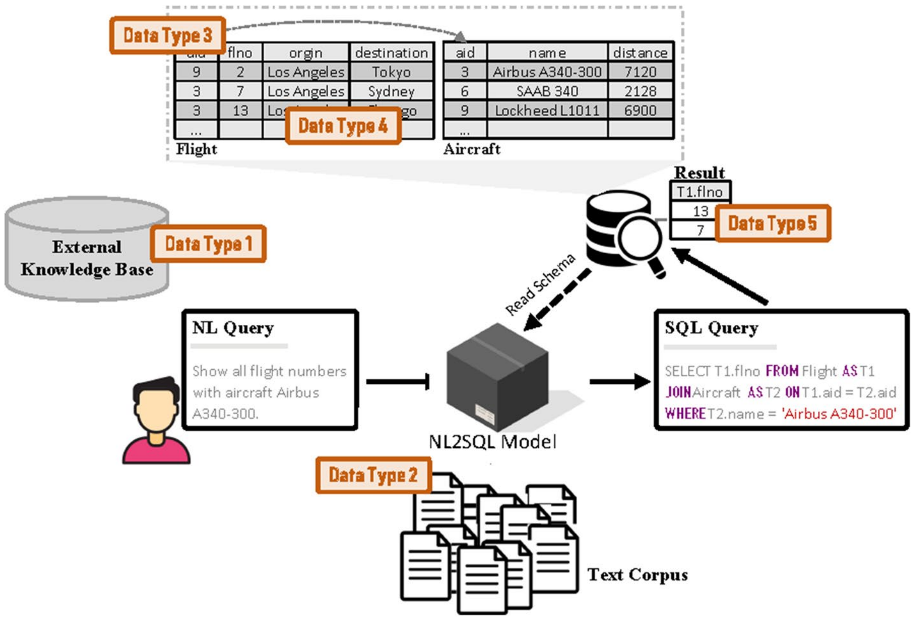
3.1 Data types involved in NL2SQL

Figure 3 illustrates the five key data types we identify as essential to the NL2SQL lifecycle. Different data sources can serve as both the foundation and catalyst for the NL2SQL process by offering diverse, high-quality inputs that enhance data pre-processing, enrich model training, and ultimately lead to more accurate SQL generation:

1. **External Knowledge** refers to supplementary information beyond the database itself, such as domain-specific ontologies, knowledge graphs, or common-sense reasoning resources. We further categorize this type of data into two main types:

   - *External Knowledge (domain)* refers to structured or curated resources specific to a particular field or industry, such as medical ontologies, legal taxonomies, or scientific knowledge bases, which provide contextual or semantic enrichment relevant to the database content.
   - *External Knowledge (LLM)* denotes information derived from large language models trained on vast corpora of general or specialized text, enabling inference, common-sense reasoning, or bridging knowledge gaps not explicitly covered by the database or domain-specific resources.

   Incorporating external knowledge sources enables more context-aware and semantically precise SQL generation, especially in complex or open-domain settings.



```
SELECT T1.flno FROM Flight AS T1
JOIN Aircraft AS T2 ON T1.aid = T2.aid
WHERE T2.name = 'Airbus A340-300'
```

Fig. 3 Data types involved in NL2SQL lifecycle

2. **Text Corpora** refers to a collection of texts, which may be either raw or annotated, and serves as a fundamental resource in the NL2SQL pipeline. These corpora are commonly used to train or fine-tune foundational translation models, enhancing their ability to comprehend and generate NL queries [13, 40]. Similarly, we also divide this type of data into two main types:
   - *Text Corpora (dataset)* refers to annotated datasets curated by humans, typically consisting of NL questions paired with corresponding SQL queries.
   - *Text Corpora (raw)* refers to unannotated text sources, such as user query logs and documentation. Despite lacking explicit labels, these corpora can be leveraged during pretraining to enhance a model's linguistic understanding and foundational capabilities.

   By leveraging diverse, high-quality text corpora, NL2SQL systems can achieve more robust generalization and improved translation accuracy across domains.
3. **Database Schema** describes the structural blueprint of a database, including table names, column names, data types, and relationships such as foreign keys. It serves as a critical reference for interpreting user queries and mapping natural language elements to their corresponding SQL components. By understanding the schema, NL2SQL systems can generate syntactically valid queries that reflect the data structure and maintain referential integrity.
4. **Database Instances** consist of the actual records stored within a database—i.e., the rows of data populating each table. These instances provide semantic grounding for model training and inference by revealing how schema elements are populated in practice. Access to instance-level data allows NL2SQL systems to learn contextual patterns, resolve ambiguity in user input, and validate query accuracy through example-based reasoning.
5. **Execution Feedback** encompasses the runtime outputs of SQL queries, including both the returned results and any error messages (or the absence of errors). This feedback loop plays a vital role in iterative model improvement by enabling techniques such as reinforcement learning, error correction, and post-hoc verification. By analyzing execution outcomes, NL2SQL systems can refine their predictions, correct invalid queries, and improve alignment with user intent over time.

3.2 NL2SQL lifecycle with data

The NL2SQL lifecycle involves a series of stages that transform NL queries into executable SQL statements, including NL query understanding, schema linking, SQL query generation, and query post-refinement. Below, we discuss the lifecycle in detail, and outline how existing solutions leverage different data sources at each stage in Table 1.

NL Query Understanding. Given an NL query, the first stage is to grasp the user's intent and identify key elements that help with the semantics understanding:

- *User Intent Understanding*. Some existing works attempt to leverage *external knowledge data* to support intent recognition, as such knowledge can help compensate for missing or implicit semantics in natural language input. Depending on how external knowledge is used, it can be incorporated through query rewriting or, more commonly, via LLMs [8, 24, 29].
- *Entity Recognition*. Entities mentioned in an NL query convey their rich semantics, such as names. In general, external knowledge data can also be used to achieve this [23].

Schema Linking. Next, the second stage involves examining the structural context—specifically, the *database schema* and *instance data*—to identify the relevant tables, columns, and cell values needed to construct the SQL query. Depending on the techniques employed, such as classifiers [7, 8, 10], GNNs [33], or prompting methods [11, 39], this process may vary significantly in complexity, accuracy, and execution efficiency. In addition to structural elements, *metadata*, such as column descriptions, data types, and foreign key constraints, can provide valuable semantic signals to improve schema linking performance, particularly for complex or ambiguous queries.

SQL Query Generation. This step typically treats the model as a black box and relies primarily on an underlying translation model to perform the task. It generally uses the *database schema* and *instance data* as input to provide the necessary context, and may also include a few-shot learning approach in LLM-based solutions [12, 23, 24, 27, 39, 40] for improved performance. In addition, to enhance the performance of a translation model, a common and effective approach is to train or fine-tune the underlying model using large-scale *text corpora* before performing inference. Such data can be sourced from websites as raw, unstructured text [13], or manually annotated by humans to provide higher-quality supervision [1, 7, 19, 40, 44].

Query Post-Refinement. The final stage is to refine the initially generated SQL query to improve its correctness

Table 1 Comparison of existing NL2SQL methods about the use of data within the NL2SQL lifecycle

Methods	NL query understanding	Schema linking	SQL query generation	Query post-refinement
RAT-SQL [19]	-	DB Schema DB Instances	DB Schema DB Instances Text Corpora (dataset)	Execution Feedback
PICARD [44]	-	DB Schema DB Instances	DB Schema DB Instances Text Corpora (dataset)	Execution Feedback
RESDSQL [7]	-	DB Schema DB Instances Text Corpora (dataset)	DB Schema DB Instances Text Corpora (dataset)	Execution Feedback
CatSQL [45]	-	DB Schema DB Instances	DB Schema DB Instances Text Corpora (dataset)	Execution Feedback
MetaSQL [10]	Text Corpora (dataset)	N/A	N/A	Text Corpora (dataset) Execution Feedback
CycleSQL [9]	N/A	N/A	N/A	DB Schema DB Instances Execution Feedback Text Corpora (dataset)
CHESS [23]	External Knowledge (llm)	DB Schema DB Instances External Knowledge (llm) External Knowledge (domain)	DB Schema x DB Instances External Knowledge (llm) External Knowledge (domain) External Knowledge (domain)	DB Schema Execution Feedback External Knowledge (llm)
Codes [13]	N/A	DB Schema DB Instances Text Corpora (dataset)	DB Schema DB Instances Text Corpora (raw) Text Corpora (dataset)	Execution Feedback
FinSQL [40]	-	DB Schema DB Instances	DB Schema DB Instances Text Corpora (dataset)	DB Schema Execution Feedback
DTS-SQL [12]	-	DB Schema DB Instances Text Corpora (dataset) External Knowledge (llm)	DB Schema DB Instances Text Corpora (dataset) External Knowledge (llm)	Execution Feedback
CHASE-SQL [24]	-	DB Schema DB Instances	DB Schema DB Instances External Knowledge (llm) External Knowledge (domain) Text Corpora (dataset)	Execution Feedback External Knowledge (llm)
PET-SQL [27]	-	DB Schema DB Instances External Knowledge (llm) External Knowledge (domain) Text Corpora (dataset)	DB Schema DB Instances External Knowledge (llm) External Knowledge (domain) Text Corpora (dataset)	Execution Feedback External Knowledge (llm)
DINSQL [39]	-	DB Schema DB Instances External Knowledge (llm)	DB Schema DB Instances External Knowledge (llm) External Knowledge (domain) Text Corpora (dataset)	External Knowledge (llm)

Table 1 (continued)

Methods	NL query understanding	Schema linking	SQL query generation	Query post-refinement
DAIL-SQL [25]	-	DB Schema	DB Schema	Execution Feedback
		DB Instances	DB Instances	
		External Knowledge (llm)	External Knowledge (llm)	
		External Knowledge (domain)	External Knowledge (domain)	
			Text Corpora (dataset)	
C3-SQL [26]	-	DB Schema	DB Schema	Execution Feedback
		DB Instances	DB Instances	
		External Knowledge (llm)	External Knowledge (llm)	
			Text Corpora (dataset)	
MAC-SQL [11]	-	DB Schema	DB Schema	External Knowledge (llm)
		DB Instances	DB Instances	Execution Feedback
		External Knowledge (llm)	External Knowledge (llm)	
			External Knowledge (domain)	
			Text Corpora (dataset)	

and alignment with user intent. A straightforward approach involves executing the query to verify its executability and evaluating the returned results to determine whether they meet the expected outcome — an approach commonly referred to as *execution feedback* [46]. Additionally, external knowledge from LLMs [23, 24, 27, 39] or training a task-specific model [9, 10], can be leveraged to assist in assessing the semantic correctness of the query.

4 NL2SQL benchmarks

With advancements in NL2SQL, a variety of datasets have emerged to address the evolving challenges, as shown in Table 2. These range from domain-specific datasets with simple queries to cross-domain datasets and further to large-scale, real-world datasets, reflecting both the progress in the field and the emergence of new challenges for NL2SQL solutions.

Single-Table, Domain-specific Datasets. Early NL2SQL datasets focused on specific domains with relatively simple SQL queries, such as ATIS [31] for flight information and Geo [14] for U.S. geographical facts. Recently, larger single-domain datasets [17, 47] have been introduced, featuring more complex databases and SQL queries tailored to specific scenarios. This shift reflects an increased emphasis on assessing the performance and practical utility of NL2SQL systems in specific domains.

Multi-Tables Cross-Domain Datasets. After the development of early single-domain datasets, the NL2SQL field shifted toward cross-domain datasets to better evaluate systems' ability to generalize across diverse SQL queries and database schemas. For instance, WikiSQL [17] was the first cross-domain dataset, drawing tables from Wikipedia spanning a wide range of topics. Spider [32], along with its various extensions [49–53], further advanced this direction by introducing complex, multi-table SQL queries and requiring models to comprehend intricate database schemas and diverse natural language question formulations across numerous domains.

Large-scale, Real-World Datasets With the emergence of LLMs, there has been a growing demand for more challenging and realistic benchmarks. As a result, several large-scale, real-world NL2SQL datasets have been introduced to better reflect practical applications and evaluate models in more demanding scenarios. Datasets such as BIRD [29], ScienceBenchmark [60], and Spider 2.0 [61] feature naturally occurring questions, real-world database schemas, and complex SQL structures, including nested queries, set operations, and joins over multiple tables. These benchmarks are designed to assess not only SQL generation accuracy but also reasoning capabilities, schema linking, and robustness to ambiguous or under-specified questions—factors that are critical for deploying NL2SQL systems in real-world environments.

5 Evaluation and analysis

This section first outlines the main evaluation metrics used in existing NL2SQL benchmarks (Sect. 5.1) and then compares the accuracy of state-of-the-art NL2SQL solutions across two popular benchmarks (Sect. 5.2).

5.1 Evaluation metrics

Evaluation metrics are essential for assessing the performance of NL2SQL systems. The following four metrics

Table 2 Statistics of NL2SQL Benchmarks

Benchmarks	Basic Info.		Database Statistics					Query Statistics			
	Year	Source	DBs	Tbls.	Tbls./DB	Cols./Tbl	Records/DB	Queries	Tbls/Query	Sels/Query	Agg
Single-Table, Domain-Specific Benchmarks											
Geo [14]	1996	annotated	1	7	7	4.14	937	880	2.22	2.19	0.92
ATIS [31]	1994	annotated	1	25	25	5.24	162,243	5418	8.39	1.79	0.22
Restaurants [30]	2000	annotated	1	3	3	4.00	19,295	560	2.43	1.17	0.35
Academic [16]	2014	annotated	1	17	17	3.12	58,249,674	196	3.48	1.04	0.54
Scholar [47]	2017	annotated	1	10	10	2.50	147,416,275	816	3.38	1.02	0.68
IMDB [48]	2017	annotated	1	17	17	3.94	40,147,386	131	2.91	1.01	0.30
Multi-Table, Cross-Domain Benchmarks											
WikiSQL [17]	2017	annotated	26531	26531	1	6.34	17	80,654	1	1	0.68
Spider [32]	2018	annotated	206	1056	5.13	5.01	8980	10,181	1.83	1.17	0.54
Spider-Syn [49]	2021	syn.	166	876	5.28	5.14	9,665	10,181	1.68	1.17	0.59
Spider-Realistic [50]	2021	syn.	166	876	5.28	5.14	9,665	10,181	1.79	1.21	0.50
Spider-DK [51]	2021	syn.	169	887	5.25	5.14	9,494	10,181	1.71	1.16	0.54
CSpider [52]	2019	syn.	206	1056	5.13	5.01	8,980	10,181	1.83	1.17	0.54
Dr.Spider [53]	2023	syn.	549	2197	4	5.54	28,460	10,181	1.81	1.19	0.52
SparC [54]	2019	annotated	166	876	5.28	5.14	9,665	15,598	1.58	1.10	0.44
CoSQL [55]	2019	annotated	166	876	5.28	5.14	9,665	15,598	1.54	1.11	0.42
KaggleDBQA [56]	2020	syn.	8	17	2.12	10.53	595,075	272	1.25	1.05	0.69
MT-TEQL [57]	2021	syn.	489,076	3279004	6.70	5.51	-	62,430	1.69	1.15	0.53
AmbiQT [58]	2023	syn.	166	876	5.28	5.14	9,665	3000	1.85	1.17	0.51
Large-Scale, Real-World Benchmarks											
BIRD [29]	2023	annotated	80	611	7.64	7.14	4,585,335	12,751	2.07	1.09	0.61
BIRD-CRITIC [59]	2023	annotated	80	611	7.64	7.14	4,585,335	600	2.07	1.09	0.61
ScienceBenchmark [60]	2023	annotated	3	50	16.67	4.98	-	5,032	1.45	1	0.24
BULL [40]	2024	annotated	3	78	26	14.96	85,631	4,966	1.22	1	0.81
Spider2.0 [61]	2025	annotated	264	6259	23.71	35.61	-	632	6.53	5.10	3.57

are commonly used to evaluate the effectiveness of existing NL2SQL solutions:

Exact Match Accuracy (EM) measures the literal equivalence between the generated SQL and the ground-truth. It checks whether the SQL clauses in both queries are structurally identical. However, because the same SQL intent can often be expressed in multiple syntactically different yet semantically equivalent ways, EM tends to underestimate the model's true performance.

Execution Accuracy (EX) evaluates correctness by comparing the execution results of the generated SQL with those of the ground-truth SQL. However, different SQL queries—potentially with distinct logic—can produce identical results, potentially leading EX to overestimate the model's true prediction accuracy.

Test-suite Accuracy (TS) [62] evaluates semantic correctness by executing predicted queries on a curated set of test databases. These databases are selected from a large pool of randomly generated instances to ensure high coverage of possible query behaviors. TS measures how accurately the predicted queries match the expected annotations across this suite, providing a strict upper bound on semantic accuracy.

Valid Efficiency Score (VES) [29] introduces this metric to measure the SQL execution efficiency of valid SQL queries. A valid SQL query is a predicted SQL whose executed results exactly match the ground truth results. The formula of VES is defined as below:

$$VES = \frac{1}{N}\sum_{n=1}^{N} I(V_n, \hat{V}_n) \cdot R(Y_n, \hat{Y}_n) \tag{2}$$

where $\hat{Y}_n$ and $\hat{V}_n$ are the predicted SQL query and its executed results, and Y_n and V_n are the ground truth SQL query and its corresponding executed results, respectively. $I(V_n, \hat{V}_n)$ is an indicator function, where:

$$I(V_n, \hat{V}_n) = \begin{cases} 1, & V_n = \hat{V}_n \\ 0, & V_n \neq \hat{V}_n \end{cases} \tag{3}$$

5.2 Analysis

We select 13 popular NL2SQL solutions and present their evaluation results on the Spider and BIRD benchmarks in Table 3. As shown, most solutions primarily focus on the EX metric. Notably, the EX scores of mainstream LLM-based methods significantly outperform those of PLM-based approaches (e.g., RESDSQL). In addition, we observe that most LLM-based approaches (e.g., DIN-SQL, DAIL-SQL) demonstrate strong generalization, achieving higher accuracy on test sets than on development sets across both benchmarks.

6 Discussion and future directions

Since the introduction of the LUNAR system [63], systems for retrieving database information have garnered increasing research attention and experienced significant growth—particularly in the area of NL2SQL in the era of LLMs. With continual improvements in model performance on benchmarks such as Spider and BIRD, there is reason for optimism, as models are becoming more advanced and capable than ever before. In this work, we build on recent experimental findings to highlight key research opportunities and open challenges from a data point of view:

NL2SQL with Data. Current methods may generate incorrect SQL results, which can be attributed to: 1) insufficient pre-understanding of the NL query, leading to ambiguous or unclear inputs; and 2) limited post-refinement that relies mainly on execution feedback without leveraging richer signals.

Enhancing Pre-Understanding. Currently, most NL2SQL approaches tend to overlook the importance of deep NL query understanding, often jumping directly to schema linking and SQL generation. However, experiences with dialogue systems [64, 65] suggest that a dedicated query-rewriting or reformulation module can significantly improve comprehension of user intent by clarifying ambiguous or context-dependent expressions before further processing. Incorporating such a module into NL2SQL pipelines could help resolve implicit references, correct grammatical issues, and normalize diverse linguistic patterns, thereby providing cleaner, more accurate inputs for the schema linking and SQL generation stages. This pre-understanding step can serve as a crucial bridge, reducing error propagation and improving overall system robustness.

Advancing Post-Refinement. Most methods primarily rely on a single data source of information, typically execution feedback, to perform initial query refinement. While execution feedback can provide valuable signals about the correctness of generated SQL queries, it is often insufficient to fully capture semantic inconsistencies or subtle errors in query logic. There is an opportunity to explore more comprehensive refinement strategies that integrate multiple types of feedback, such as user interaction signals [66], semantic validation against schema constraints, and cross-checking with alternative query formulations. Additionally, leveraging iterative refinement loops [9] with LLM-based reasoning or incorporating external knowledge sources could further enhance the accuracy and reliability of post-generation corrections. By adopting more sophisticated and holistic refinement mechanisms, NL2SQL systems can better handle complex queries and ambiguous user

Table 3 Accuracies of state-of-the-art NL2SQL solutions over two popular benchmarks

Approaches	Spider					BIRD		
	Dev			Test		Dev	Test	
	EM (%)	EX (%)	TS (%)	EM (%)	EX (%)	EX (%)	EX (%)	VES (%)
RESDSQL+T5-3B [7]	76.0	79.4	73.5	70.2	78.4	-	-	-
C3+GPT-3.5-Turbo [26]	43.1	81.8	72.1	-	82.3	-	-	-
DAIL-SQL+GPT-4 [25]	22.1	72.3	-	-	86.6	54.76	57.41	54.02
CHESS [23]	-	-	-	-	87.2	68.31	71.10	62.77
PURPLE+GPT-4o [8]	80.5	87.8	83.3	-	87.2	62.97	64.51	65.62
MetaSQL+LGESQL [10, 22]	77.4	42.0	-	72.3	55.7	-	-	-
CycleSQL+GPT-3.5-Turbo [9]	49.2	77.8	66.2	50.6	75.1	-	-	-
CodeS-15B [13]	-	84.9	79.4	-	-	58.47	60.37	56.73
DINSQL+GPT-4 [39]	60.1	82.80	74.2	-	85.30	50.72	55.90	53.07
MAC-SQL+GPT-4 [11]	-	86.75	-	-	82.80	57.56	59.59	57.60
SuperSQL [46]	-	-	-	87.0	-	-	62.66	61.99
CHASE+Gemini 1.5 [24]	-	-	-	-	87.6	73.01	73.0	-
DTS-SQL+DeepSeek-7B [12]	79.1	85.5	-	73.7	84.4	-	-	-

intents, ultimately improving end-to-end performance and user satisfaction.

Bridging NL2SQL and NL2Code. NL2SQL can be viewed as a specialized subfield within the broader domain of assist programming [67, 68], namely NL2Code. Insights and techniques from NL2Code research offer valuable opportunities for advancing NL2SQL systems. For example, integrating similar multi-step reasoning processes, code synthesis strategies, and debugging paradigms from automated programming [69] could enhance NL2SQL robustness, especially for complex queries involving nested sub-queries. Furthermore, NL2Code often explores interactive code generation and human-in-the-loop refinement, which aligns with the earlier-mentioned idea of leveraging user feedback and iterative refinement in NL2SQL pipelines. Thus, bridging these two areas can foster cross-pollination of methodologies, such as applying program synthesis techniques, formal verification, and execution-based debugging from general code generation to improve NL2SQL accuracy and reliability.

Noise-Inspired Robustness. Recent advances in positive-incentive noise and related studies [70–75] show that structured noise can enhance generalization and stability. Similar ideas could be applied to NL2SQL—injecting beneficial noise into representations or decoding processes to improve resilience against ambiguous user inputs and better adapt to evolving database schemas.

System-Level Perspectives. [76] highlights lifecycle orchestration across heterogeneous AI components. Viewing NL2SQL through this lens suggests integrating understanding, generation, and feedback as a continuous flow for adaptive and scalable query systems. Moreover, multimodal alignment principles from vision-language studies [77, 78] can inform more interpretable, human-aligned database interfaces.

Beyond Accuracy. Generating correct SQL is not enough: efficiency and dialect compatibility are also crucial for real-world deployment. Different database systems support varying SQL dialects and optimize queries differently. A valid but inefficient query can cause significant performance issues. To address this, NL2SQL systems should incorporate metadata (e.g., indexes and keys) and statistics (e.g., column cardinality and histograms) to guide query generation. Leveraging such signals can help produce queries that are both executable and optimized for the target engine, drawing from techniques in learned query optimization and cost-based reformulation.

7 Conclusion

In this paper, we provided a comprehensive review of NL2SQL from a lifecycle perspective, with a focus on the role of data. We formally defined the task, identified key challenges, and categorized the major data types involved. We then analyzed how these data sources are utilized throughout the NL2SQL lifecycle, including NL understanding, schema linking, SQL query generation, and query post-refinement. Furthermore, we examined existing benchmarks and evaluation metrics, highlighting their characteristics and common errors. Finally, we discussed promising research directions and open challenges in the field.

Abbreviations

NL2SQL	Natural Language to SQL Translation
NL2Code	Natural Language to Code
NL	Natural Language
NLP	Natural Language Processing
LLM	Large Language Model
PLM	Pre-trained Language Model
Seq2seq	Sequence to Sequence
GNN	Graph Neural Network
DB	Database
EM	Exact Match Accuracy
EX	Execution Accuracy
TS	Test-suite Accuracy
VES	Valid Efficiency Score

Acknowledgements
Not applicable.

Code availability
Not applicable.

Authors' contributions
This paper was a collaborative effort. Yuankai Fan conceived and developed the study. Qizhen Weng led the writing and analysis of the manuscript. X. Sean Wang contributed and guided the research direction and provided critical revisions to ensure the manuscript's quality. Yin Chen was the project administrator. All authors critically reviewed draft versions of the manuscripts, and all authors gave final approval of the version to be submitted.

Funding
Not applicable.

Data availability
All data used in this paper is publicly available. No new data was used for the research described in the paper.

Declarations

Competing interests
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work in this paper.

Received: 31 July 2025 Revised: 5 November 2025 Accepted: 12 November 2025
Published online: 26 November 2025

References

1. Y. Fan, Z. He, T. Ren, D. Guo, L. Chen, R. Zhu, G. Chen, Y. Jing, K. Zhang, X.S. Wang, in *ICDE*, Gar: A generate-and-rank approach for natural language to SQL translation (2023), pp.110–122
2. Y. Fan, T. Ren, Z. He, X.S. Wang, Y. Zhang, X. Li, in *ICDE*, Gensql: A generative natural language interface to database systems (2023), pp.3603–3606

3. Y. Fan, C. Huang, T. Ren, Z. He, X.S. Wang, X. Wu, Y. Wang, J. Li, Y. Yang, Gar++: Natural language to SQL translation with efficient generate-and-rank. Lecture Notes in Computer Science. In: *APWeb-WAIM*, vol. 14963 (Springer, 2024), pp. 411–427. https://doi.org/10.1007/978-981-97-7238-4_26
4. Y. Fan, T. Ren, C. Huang, B. Zheng, Y. Jing, Z. He, J. Li, J. Li, A confidence-based knowledge integration framework for cross-domain table question answering. Knowl.-Based Syst. **306**, 112718 (2024). https://doi.org/10.1016/j.knosys.2024.112718
5. Z. Gu, J. Fan, N. Tang, S. Zhang, Y. Zhang, Z. Chen, L. Cao, G. Li, S. Madden, X. Du, Interleaving pre-trained language models and large language models for zero-shot NL2SQL generation. CoRR (2023). arXiv:2306.08891
6. R. Sun, S.Ö. Arik, H. Nakhost, H. Dai, R. Sinha, P. Yin, T. Pfister, Sql-palm: Improved large language model adaptation for text-to-sql. CoRR (2023). arXiv:2306.00739
7. H. Li, J. Zhang, C. Li, H. Chen, RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql (2023). In: *AAAI*, pp. 13,067–13,075
8. T. Ren, Y. Fan, Z. He, R. Huang, J. Dai, C. Huang, Y. Jing, K. Zhang, Y. Yang, X.S. Wang, PURPLE: making a large language model a better SQL writer. In: *ICDE*, pp. 15–28. (2024)
9. Y. Fan, T. Ren, C. Huang, Z. He, X.S. Wang, Grounding natural language to sql translation with data-based selfexplanations. In: *ICDE*, pp. 29–42. (2025)
10. Y. Fan, Z. He, T. Ren, C. Huang, Y. Jing, K. Zhang, X.S. Wang, Metasql: A generatethen-rank framework for natural language to SQL translation. In: *ICDE*, pp. 1765–1778. (2024)
11. B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L. Chai, Z. Yan, Q. Zhang, D. Yin, X. Sun, Z. Li, MAC-SQL: A multi-agent collaborative framework for text-to-sql. In: *COLING*, pp. 540–557. (2025)
12. M. Pourreza, D. Rafiei, DTS-SQL: decomposed text-to-sql with small large language models. In: *EMNLP*, pp. 8212–8220. (2024)
13. H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, H. Chen, Codes: towards building open-source language models for text-to-sql. Proc. ACM Manag. Data **2**(3), 127 (2024)
14. J.M. Zelle, R.J. Mooney, Learning to parse database queries using inductive logic programming. In: *AAAI* (1996)
15. A. Simitsis, G. Koutrika, Y.E. Ioannidis, Précis: from unstructured keywords as queries to structured databases as answers. PVLDB **1**, 117–149 (2008)
16. F. Li, H.V. Jagadish, Constructing an interactive natural language interface for relational databases. PVLDB **1**, 73–84 (2014)
17. V. Zhong, C. Xiong, R. Socher, Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR (2017)
18. X. Xu, C. Liu, D. Song, Sqlnet: Generating structured queries from natural language without reinforcement learning. CoRR (2017)
19. B. Wang, R. Shin, X. Liu, O. Polozov, M. Richardson, RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. In: *ACL*, pp. 7567–7578 (2020)
20. J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J. Lou, T. Liu, D. Zhang, Towards complex text-tosql in cross-domain database with intermediate representation. In: *ACL*, pp. 4524–4535 (2019)
21. O. Rubin, J. Berant, Smbop: Semiautoregressive bottom-up semantic parsing. In: *NAACL-HLT*, pp. 311–324 (2021)
22. R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, K. Yu, LGESQL: line graph enhanced text-to-sql model with mixed local and nonlocal relations. In: *ACL/IJCNLP*, pp. 2541–2555 (2021)
23. S. Talaei, M. Pourreza, Y. Chang, A. Mirhoseini, A. Saberi, CHESS: contextual harnessing for efficient SQL synthesis. CoRR (2024). arXiv:2405.16755
24. M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G.T. Kakkar, Y. Gan, A. Saberi, F. Ozcan, S.Ö. Arik, CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-sql. In: *ICLR* (2025)
25. D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, J. Zhou, Text-to-sql empowered by large language models: A benchmark evaluation. CoRR (2023). arXiv:2308.15363
26. X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, L. Chen, J. Lin, D. Lou, C3: zero-shot text-to-sql with chatgpt. CoRR (2023)
27. Z. Li, X. Wang, J. Zhao, S. Yang, G. Du, X. Hu, B. Zhang, Y. Ye, Z. Li, R. Zhao, Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency (2024). arXiv preprint arXiv:2403.09732
28. G. Katsogiannis-Meimarakis, G. Koutrika, A deep dive into deep learning approaches for text-to-sql systems. In: *SIGMOD*, pp. 2846–2851. (2021)
29. J. Li, B. Hui, G. Qu, B. Li, J. Yang, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, N. Huo, C. Ma, K.C.C. Chang, F. Huang, R. Cheng, Y. Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls (2023)
30. L.R. Tang, R.J. Mooney, Intergrating statistical and relational learning for semantic parsing. In: *SIGDAT*, pp. 133–141. (2000)
31. D.A. Dahl, M. Bates, M. Brown, W.M. Fisher, K. Hunicke-Smith, D.S. Pallett, C. Pao, A.I. Rudnicky, E. Shriberg, Expanding the scope of the ATIS task: The ATIS-3 corpus. In: *Human Language Technology*. (1994)
32. T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, D.R. Radev, Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: *EMNLP* (2018)
33. B. Bogin, J. Berant, M. Gardner, Representing schema structure with graph neural networks for text-to-sql parsing. In: *ACL*, pp. 4560–4565 (2019)
34. J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding. In: *NAACL* (2019)
35. X.V. Lin, R. Socher, C. Xiong, Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In: *EMNLP*, pp. 4870–4888 (2020)
36. T. Yu, C. Wu, X.V. Lin, B. Wang, Y.C. Tan, X. Yang, D.R. Radev, R. Socher, C. Xiong, Grappa: Grammar-augmentedpre-training for table semantic parsing. In: *ICLR* (2021)
37. T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners. In: *NeurIPS* (2020)
38. OpenAI, GPT-4 technical report. CoRR (2023)
39. M. Pourreza, D. Rafiei, DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In: *NeurIPS* (2023)
40. C. Zhang, Y. Mao, Y. Fan, Y. Mi, Y. Gao, L. Chen, D. Lou, J. Lin, Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. In: *SIGMOD*, pp. 93–105.(2024)
41. 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, B. Xiang, Importance of synthesizing highquality data for text-to-sql parsing. In: *ACL*, pp. 1327–1343. (2023). https://doi.org/10.18653/V1/2023.FINDINGS-ACL.86
42. Y. Guo, D. Jin, S. Ye, S. Chen, J. Jianyang, X. Tan, Synthesizing reliable and diverse data to enhance text-to-sql reasoning in llms. In: *ACL*, pp. 8441–8452. (2025)
43. H. Li, S. Wu, X. Zhang, X. Huang, J. Zhang, F. Jiang, S. Wang, T. Zhang, J. Chen, R. Shi, H. Chen, C. Li, Omnisql: Synthesizing high-quality text-to-sql data at scale. CoRR (2025). arXiv:2503.02240
44. T. Scholak, N. Schucher, D. Bahdanau, PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In: *EMNLP*, pp. 9895–9901 (2021)
45. H. Fu, C. Liu, B. Wu, F. Li, J. Tan, J. Sun, Catsql: towards real world natural language to SQL applications. Proc. VLDB Endow. **16**(6), 1534–1547 (2023)
46. B. Li, Y. Luo, C. Chai, G. Li, N. Tang, The dawn of natural language to SQL: are we fully ready? Proc. VLDB Endow. **17**(11), 3318–3331 (2024)
47. S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, L. Zettlemoyer, Learning a neural semantic parser from user feedback. In: *ACL*, pp. 963–973. (2017)
48. N. Yaghmazadeh, Y. Wang, I. Dillig, T. Dillig, Sqlizer: query synthesis from natural language. Proc. ACM Program. Lang. **1**, 63:1–63:26 (2017)
49. Y. Gan, X. Chen, Q. Huang, M. Purver, J.R. Woodward, J. Xie, P. Huang, Towards robustness of text-to-sql models against synonym substitution. In: *ACL*, pp. 2505–2515 (2021)
50. X. Deng, A.H. Awadallah, C. Meek, O. Polozov, H. Sun, M. Richardson, Structuregrounded pretraining for text-to-sql. In: *NAACL-HLT*, pp. 1337–1350 (2021)
51. Y. Gan, X. Chen, M. Purver, Exploring underexplored limitations of cross-domain text-to-sql generalization. In: *EMNLP*, pp. 8926–8931 (2021)
52. Q. Min, Y. Shi, Y. Zhang, A pilot study for chinese SQL semantic parsing. In: *EMNLP*, pp. 3650–3656. (2019)
53. 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, B. Xiang, Dr.spider: A diagnostic evaluation benchmark towards text-to-sql robustness. In: *ICLR* (2023)

54. 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, D.R. Radev, Sparc: Cross-domain semantic parsing in context. In: *ACL*, pp. 4511–4523. (2019)
55. 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, D.R. Radev, Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In: *EMNLP*, pp. 1962–1979. (2019)
56. C. Lee, O. Polozov, M. Richardson, Kaggledbqa: Realistic evaluation of text-to-sql parsers. In: *ACL*, pp. 2261–2273. (2021)
57. P. Ma, S. Wang, Mt-teql: evaluating and augmenting neural NLIDB on real-world linguistic and schema variations. Proc. VLDB Endow. **15**(3), 569–582 (2021)
58. A. Bhaskar, T. Tomar, A. Sathe, S. Sarawagi, Benchmarking and improving text-to-sql generation under ambiguity. In: *EMNLP*, pp. 7053–7074. (2023)
59. J. Li, X. Li, G. Qu, P. Jacobsson, B. Qin, B. Hui, S. Si, N. Huo, X. Xu, Y. Zhang, Z. Tang, Y. Li, F. Widjaja, X. Zhu, F. Zhou, Y. Huang, Y. Papakonstantinou, F. Ozcan, C. Ma, R. Cheng, SWE-SQL: illuminating LLM pathways to solve user SQL issues in real-world applications. CoRR (2025). arXiv:2506.18951
60. Y. Zhang, J. Deriu, G. Katsogiannis-Meimarakis, C. Kosten, G. Koutrika, K. Stockinger, Sciencebenchmark: a complex real-world benchmark for evaluating natural language to SQL systems. Proc. VLDB Endow. **17**(4), 685–698 (2023)
61. F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, V. Zhong, C. Xiong, R. Sun, Q. Liu, S. Wang, T. Yu, Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In: *ICLR* (2025)
62. R. Zhong, T. Yu, D. Klein, Semantic evaluation for text-to-sql with distilled test suites. In: *EMNLP*, pp. 396–411 (2020)
63. W.A. Woods, Progress in natural language understanding: an application to lunar geology. In: *American Federation of Information Processing Societies*. AFIPS Conference Proceedings, vol. 42, pp. 441–450. (1973). https://doi.org/10.1145/1499586.1499695
64. S. Bao, H. He, F. Wang, H. Wu, H. Wang, W. Wu, Z. Guo, Z. Liu, X. Xu, PLATO-2: towards building an open-domain chatbot via curriculum learning. In: *ACL/IJCNLP. Findings of ACL*, vol. ACL/IJCNLP 2021, pp. 2513–2525. (2021). https://doi.org/10.18653/V1/2021.FINDINGS-ACL.222
65. J. Ni, T. Young, V. Pandelea, F. Xue, E. Cambria, Recent advances in deep learning based dialogue systems: a systematic survey. Artif. Intell. Rev. **56**(4), 3055–3155 (2023). https://doi.org/10.1007/S10462-022-10248-8
66. Y. Fan, T. Ren, D. Guo, Z. Zhao, Z. He, X.S. Wang, Y. Wang, T. Sui, An integrated interactive framework for natural language to SQL translation. In: *WISE. Lecture Notes in Computer Science*, vol. 14306, pp. 643–658. (2023). https://doi.org/10.1007/978-981-99-7254-8_50
67. M. Chen, J. Tworek, H. Jun, Q. Yuan, H.P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F.P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W.H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A.N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, W. Zaremba, Evaluating large language models trained on code. CoRR (2021). arXiv:2107.03374
68. H. Jiang, Q. Liu, R. Li, S. Ye, S. Wang, Cursorcore: Assist programming through aligning anything. CoRR (2024). arXiv:2410.07002
69. J. Jiang, F. Wang, J. Shen, S. Kim, S. Kim, A survey on large language models for code generation. CoRR (2024). arXiv:2406.00515
70. X. Li, Positive-incentive noise. IEEE Trans. Neural Netw. Learn. Syst. **35**(6), 8708–8714 (2024)
71. H. Zhang, S. Huang, Y. Guo, X. Li, Variational positive-incentive noise: how noise benefits models. IEEE Trans. Pattern Anal. Mach. Intell. (2025). https://doi.org/10.1109/TPAMI.2025.3575295
72. H. Zhang, Y. Xu, S. Huang, X. Li, Data augmentation of contrastive learning is estimating positive-incentive noise (2024). arXiv preprint arXiv:2408.09929
73. S. Huang, H. Zhang, X. Li, Enhance visionlanguage alignment with noise. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, pp. 17449–17457 (2025)
74. S. Huang, Y. Xu, H. Zhang, X. Li, Learn beneficial noise as graph augmentation. In: *Proceedings of the 42nd International Conference on Machine Learning (ICML)* (2025)
75. K. Jiang, Z. Shi, D. Zhang, H. Zhang, X. Li, Mixture of noise for pre-trained model-based class-incremental learning (2025). arXiv preprint arXiv:2509.16738
76. H. An, W. Hu, S. Huang, S. Huang, R. Li, Y. Liang, J. Shao, Y. Song, Z. Wang, C. Yuan, C. Zhang, H. Zhang, W. Zhuang, X. Li, Ai flow: Perspectives, scenarios, and approaches (2025). arXiv preprint arXiv:2506.12479
77. J. Han, D. Zhang, S. Wen, L. Guo, T. Liu, X. Li, Two-stage learning to predict human eye fixations via SDAEs. IEEE Trans. Cybern. **46**(2), 487–498 (2015)
78. D. Tao, M. Song, X. Li, J. Shen, J. Sun, X. Wu, C. Faloutsos, S.J. Maybank, Bayesian tensor approach for 3-d face modeling. IEEE Trans. Circ. Syst. Video Technol. **18**(10), 1397–1410 (2008)

Publisher's Note