NL2SQLBench: A Modular Benchmarking Framework for LLM-Enabled NL2SQL Solutions

Shizheng Hou
National University of Singapore
shizheng_hou@u.nus.edu

Wenqi Pei
National University of Singapore
wenqi_pei@u.nus.edu

Nuo Chen
National University of Singapore
nuochen@comp.nus.edu.sg

Quang-Trung Ta
National University of Singapore
taqt@nus.edu.sg

Peng Lu*
Zhejiang University
peng.lu@zju.edu.cn

Beng Chin Ooi
Zhejiang University
ooibc@zju.edu.cn

ABSTRACT

Natural Language to SQL (NL2SQL) technology empowers non-expert users to query relational databases without requiring SQL expertise. While large language models (LLMs) have greatly improved NL2SQL algorithms, their rapid development outpaces systematic evaluation, leaving a critical gap in understanding their effectiveness, efficiency, and limitations. To this end, we present **NL2SQLBench**, the first modular evaluation and benchmarking framework for LLM-enabled NL2SQL approaches. Specifically, we dissect NL2SQL systems into three core modules: Schema Selection, Candidate Generation, and Query Revision. For each module, we comprehensively review existing strategies and propose novel fine-grained metrics that systematically quantify module-level effectiveness and efficiency. We further implement these metrics in a flexible multi-agent framework, allowing configurable benchmarking across diverse NL2SQL approaches. Leveraging **NL2SQLBench**, we rigorously evaluate ten representative open-source methods on two datasets, the BIRD development set and the ScienceBenchmark development set, using two LLMs, DeepSeek-V3 and GPT-4o mini. We systematically assess each approach across the three core modules and evaluate multiple critical performance dimensions. Our evaluation reveals significant gaps in existing NL2SQL methods, highlighting not only substantial room for accuracy improvements but also the significant computational inefficiency, which severely hampers real-world adoption. Furthermore, our analysis identifies critical shortcomings in current benchmark datasets and evaluation rules, emphasizing issues such as inaccurate gold SQL annotations and limitations in existing evaluation rules. By synthesizing these detailed insights into a unified, transparent, and reproducible benchmarking, our study not only establishes a clear reference point for fair comparison across approaches but also serves as essential guidance for future targeted innovation in NL2SQL technology, thus advancing the practical deployment and real-world applicability of NL2SQL technologies.

PVLDB Reference Format:
Shizheng Hou, Wenqi Pei, Nuo Chen, Quang-Trung Ta, Peng Lu, and Beng Chin Ooi. **NL2SQLBench**: A Modular Benchmarking Framework for LLM-Enabled NL2SQL Solutions. PVLDB, 19(5): 1001 - 1015, 2026.
doi:10.14778/3796195.3796211

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 5 ISSN 2150-8097.
doi:10.14778/3796195.3796211

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HOU-SZ/NL2SQLBench.

1 INTRODUCTION

Natural Language to SQL (NL2SQL) is a pivotal technology that empowers non-expert users to query relational databases using natural language (NL) questions, without requiring any SQL expertise. This technology holds significant value as it democratizes access to data, empowering users to engage with complex database systems and supporting diverse applications across various business domains [32, 39, 55, 58, 65, 69, 74]. Concurrently, as noted by Liu et al. [34], the provision of NL2SQL solutions has shifted from a conceptual idea to an essential strategy among database vendors. Consequently, the question of how to effectively implement such solutions has become an actively discussed topic [1, 12].

Recent advancements in Large Language Models (LLMs) [20, 42, 77] have introduced a transformative paradigm for NL2SQL tasks, offering new opportunities for enhanced performance and adaptability. Numerous LLM-based approaches [2, 3, 10, 11, 14, 15, 21, 27, 29, 31, 36–38, 47, 49, 51, 53, 54, 60, 72, 76] have been proposed, establishing themselves as the most prominent solutions in the NL2SQL landscape.

As articulated by Liu et al. [34], these systems typically consist of three core modules: (1) *Schema Selection*: select the most relevant tables and columns from the databases; (2) *Candidate Generation*: generate candidate SQL queries based on the NL queries; (3) *Query Revision*: refine the candidate SQL queries to generate the final SQL queries. These modules form the fundamental structure of most modern NL2SQL systems, as depicted in the top half of Figure 1.

The primary evaluation metrics currently employed in existing NL2SQL benchmarks [22, 28, 71], typically employ three metrics, namely *Exact Match (EM)*, *Execution Accuracy (EX)*, and *Valid Efficiency Score (VES)* that only concern the overall correctness and execution efficiency of the finally generated SQL. However, such metrics do not assess the effectiveness and efficiency of each individual core module, hindering deeper insights into where improvements are needed most urgently.

Furthermore, although an increasing number of solutions continue to shine on NL2SQL leaderboards [22, 28, 71], the computational costs and latency incurred by LLMs remain overlooked.

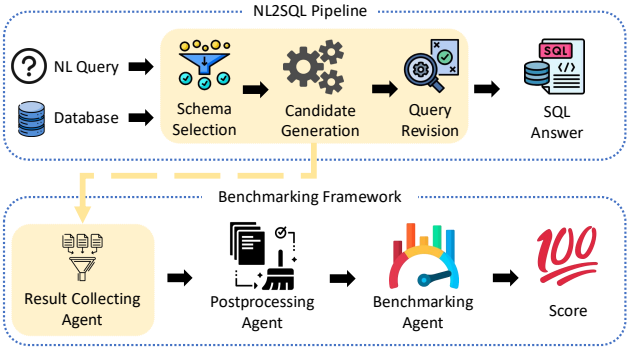


Figure 1: Overview of LLM-enabled NL2SQL approaches and our benchmarking framework.

Additionally, variability in evaluation settings across studies complicates fair comparisons between different strategies for each module. There is no unified framework to perform a holistic comparison of the individual modules under the same experimental setting.

Therefore, the rapid development of LLM-based NL2SQL methods has outpaced systematic evaluation, leaving a critical gap in understanding their effectiveness and efficiency. These limitations highlight two urgent and interconnected research questions: (1) *How can we systematically measure the effectiveness and efficiency of individual NL2SQL modules?*, and (2) *How can we establish a consistent, reproducible framework to evaluate these modules across different methods fairly?* Answering these questions is crucial for understanding the performance of techniques used in each module, which ultimately guides the development of more accurate and efficient NL2SQL algorithms.

Our work. To tackle these challenges, we present **NL2SQLBench**, the first modularized benchmarking framework specifically designed for systematic evaluation of the core modules *Schema Selection*, *Candidate Generation*, and *Query Revision* of LLM-enabled NL2SQL solutions. We first review comprehensively different techniques implemented in each module by existing NL2SQL systems. To facilitate the modularized evaluation, we propose new evaluation metrics for each module, enabling a holistic assessment of the effectiveness and efficiency of these techniques. Then, we develop a flexible, multi-agent benchmarking framework consisting of three agents: *Result Collecting*, *Result Postprocessing*, and *Benchmarking* agent. These agents are designed to be modular and configurable, allowing compatibility with diverse NL2SQL approaches.

We apply our framework to evaluate ten representative and open-sourced LLM-based NL2SQL approaches from the BIRD leaderboard[1] [28], using two datasets, the BIRD [28] and the ScienceBenchmark [75] datasets, and two LLMs, DeepSeek-V3 [8] and GPT-4o mini [46]. Then, we conduct a comprehensive analysis of their performance. Our comprehensive benchmarking analysis reveals that apart from the large room for improving accuracy at the modular level, computational efficiency is another obstacle to industrial adoption. Furthermore, our findings highlight previously underappreciated challenges regarding benchmark dataset quality and evaluation rules, areas that warrant immediate attention. Beyond just identifying these challenges, we further synthesize our insights into a series of practical guides and usability analyses (e.g., Sec. 4.2.4, 4.7), offering actionable guidance for developers to navigate the accuracy-efficiency trade-offs in real-world system development.

Contributions. We summarize our contributions as follows.

- We comprehensively review the three core modules of LLM-enabled NL2SQL solutions, namely *Schema Selection*, *Candidate Generation*, and *Query Revision*, on their details, representative implementations.
- We propose a set of new, fine-grained evaluation metrics specifically tailored for systematically assessing the effectiveness and efficiency of each individual module. Unlike existing overall metrics, our proposed metrics enable precise, detailed analyses of performance across multiple dimensions.
- We develop **NL2SQLBench**, the first modular, multi-agent benchmarking framework for evaluating core modules. Our framework comprises three seamlessly integrated agents—*Result Collecting*, *Result Postprocessing*, *Benchmarking*—each designed with flexibility in mind, ensuring broad applicability across diverse NL2SQL methods.
- Leveraging **NL2SQLBench**, we comprehensively benchmark ten representative open-source NL2SQL approaches from the BIRD leaderboard. Our evaluation reveals substantial room for improvement in both accuracy and computational efficiency, warranting further study and providing valuable insights for researchers aiming to advance NL2SQL methodologies.
- Our in-depth analysis uncovers previously unnoticed yet critical limitations in the quality of existing datasets and evaluation rules. Our findings strongly advocate for the creation of more robust, reliable, and flexible benchmark datasets and evaluation methods in future research.
- We distill the benchmarking results into practical guides and usability analyses, providing concrete recommendations and a systematic workflow to help practitioners effectively diagnose bottlenecks, navigate design trade-offs, and iteratively improve NL2SQL systems.

NL2SQLBench has been developed as a part of NeurDB [45, 78] for supporting friendly and yet effective query interface to the AI-powered autonomous data system.

2 COMPREHENSIVE REVIEW OF LLM-ENABLED NL2SQL SOLUTIONS

2.1 NL2SQL Task Formulation

An NL2SQL task involves translating a natural language query into a corresponding SQL statement that can be executed on a relational database. Specifically, given a natural language query Q on a database D, the task is aimed to produce a SQL statement S based on Q and D. Formally, the task can be defined as: $S = f(Q, D)$.

Most LLM-enabled NL2SQL approaches follow a three-stage architecture comprising three core modules: *Schema Selection*, *Candidate Generation*, and *Query Revision*, which correspond to the three phrases *pre-processing*, query generation, and post-processing [34, 57], as depicted in Figure 1. Since these three modules all leverage LLMs to perform their functionalities, they need to consider various aspects such as how to design proper prompts, select few-shot examples, and decide the context length to feed into the LLMs.

[1] https://bird-bench.github.io/

In the following, we comprehensively review how these modules are implemented in existing systems.

2.2 Schema Selection Module

Leveraging LLMs to generate SQL queries from an NL question requires constructing a prompt that incorporates the question, database schema, and task-specific instructions. However, the limited context window of LLMs poses challenges in accommodating large schemas [12, 32]. Furthermore, lengthy prompts increase cost and degrade performance [33]. Hence, the *Schema Selection* module, which is also known as schema linking [3, 4, 17, 23, 26, 34, 49, 61], is implemented by many solutions to extract the most relevant tables and columns. We describe representative strategies employed in existing works below and summarize them in Table 1.

Table 1: Classification of Schema Selection strategies.

Approach	Few-shot CoT	Multi-Stage Pruning	Preliminary SQL
C3-SQL [10]	✗	✓	✗
DIN-SQL [49]	✓	✗	✗
MAC-SQL [60]	✓	✗	✗
$\text{CHESS}_{(\text{IR,SS,CG})}$ [59]	✓	✓	✗
TA-SQL [53]	✗	✗	✓
OpenSearch-SQL [67]	✓	✓	✗
RSL-SQL [4]	✓	✗	✓
PET-SQL [31]	✓	✗	✓
GSR [16]	✗	✗	✓

Few-Shot Chain-of-Thought (CoT). This strategy utilizes the few-shot in-context learning [9, 30] capabilities of LLMs supplemented by CoT prompting [6, 64] to identify relevant tables and columns for NL queries, enabling LLMs to generalize without extensive task-specific training. For instance, MAC-SQL [60] and DIN-SQL [49] select demonstrations that map NL queries to target schemas. By guiding the LLM through a step-by-step process, the CoT method helps the model better interpret the NL query and effectively select relevant tables and columns.

Multi-Stage Schema Pruning. This strategy progressively narrows the database schema in multiple steps to ensure that only the necessary schemas are considered. For instance, $\text{CHESS}_{(\text{IR,SS,CG})}$ [59] leverages locality-sensitive hashing and vector retrieval to efficiently identify relevant database values and catalogs, followed by a three-stage pruning process: broadly columns filtering, followed by selecting the most relevant table, and finally picking necessary columns. OpenSearch-SQL [67] employs a similar way: information retrieval and column filtering - but chooses to recall relevant schema by leveraging both LLMs and vector retrieval.

Preliminary-SQL Enhanced. Instead of explicitly instructing LLMs on schema selection, this method begins by prompting the LLMs to generate a preliminary SQL query based on the natural language query. Once such a query is generated, relevant schema elements, such as tables, columns, and relationships, are extracted from it. TA-SQL [53] is a notable example of employing this strategy. It first generates a dummy SQL query and then extracts schema entities. PET-SQL [31] and GSR [16] also adopt similar methods.

Hybrid Approaches. Multiple strategies can be combined in a hybrid approach to improve their effectiveness. For example, RSL-SQL [4] combines zero-shot schema filtering and a preliminary SQL-based method, thus achieving a bidirectional schema linking design. $\text{CHESS}_{(\text{IR,SS,CG})}$ [59] also employs a hybrid approach that combines few-shot CoT and schema pruning.

Horizontal Comparison. Few-shot CoT has low overhead and adapts quickly, but is sensitive to exemplar coverage and phrasing. Multi-stage pruning performs well and is robust under ambiguity, but it is very token-intensive and increases latency. Preliminary-SQL is token-efficient and precise when the draft is reliable but degrades with draft errors or drift.

2.3 Candidate Generation Module

The Candidate Generation module is tasked with converting NL queries into candidate SQL statements. This module directly interprets user intent and produces the corresponding SQL queries that align with the structure and constraints of the database schema. We describe the representative strategies for this module as follows and present the classification of existing works in Table 2.

Few-Shot CoT. By explicitly incorporating intermediate reasoning steps in few-shot NL2SQL generation examples, few-shot CoT methods help LLMs process and solve complex logical processes in NL2SQL. For example, DIN-SQL [49] employs human-designed CoT frameworks for different types of NL questions.

Query Classification. This strategy classifies NL queries into different classes according to their complexity and uses different prompts for each class. DIN-SQL [49] is a representative example that employs a classification strategy to generate SQL candidates.

Query Decomposition. The decomposition-based strategy extends the CoT approach by decomposing a problem into smaller, manageable components [68]. The decomposer agent introduced by MAC-SQL [60] breaks the query into a series of intermediate steps, such as sub-questions, and generates corresponding sub-queries for each step before generating the final SQL.

Intermediate Representation. To bridge the gap between NL and SQL queries, intermediate representations (IRs) [13, 17, 26], such as Pandas-like or SQL-like codes, have been introduced to facilitate the generation of SQL queries. TA-SQL [53] employs Pandas-like code as an IR, DIN-SQL adopts the IR from NatSQL [13], while OpenSearch-SQL invents an SQL-like language to encourage LLMs to focus more on logic before generating final SQL queries.

Multiple Candidate Generation. Some recent works choose to increase LLM calls or sampling numbers and generate multiple candidates before selecting the final answers among them. This strategy has been implemented in studies such as C3-SQL [10], CHESS [59], MCS-SQL [21] and OpenSearch-SQL [67].

Hybrid Approaches. Many works mentioned above also adopt multiple strategies to enhance performance.

Horizontal Comparison. Few-shot prompting is efficient but brittle to template and phrasing shift. Query classification improves alignment at modest cost but risks misrouting. Decomposition increases controllability and traceability but adds steps and propagates errors. Intermediate representations stabilize structure and reduce surface variance but may under-express rare constructs and

add translation burden. Multi-candidate generation broadens coverage and hedges uncertainty but raises computation and depends on reliable selection.

Table 2: Classification of Candidate Generation strategies.

Approach	Few-shot CoT	Classification	Decomposition	Inter Represent	Multi Candidate
C3-SQL [10]	✗	✗	✗	✗	✓
DIN-SQL [49]	✓	✓	✗	NatSQL	✗
MAC-SQL [60]	✓	✗	✓	✗	✗
$\text{CHESS}_{(\text{IR,SS,CG})}$ [59]	Zero-Shot	✗	✗	✗	✗
$\text{CHESS}_{(\text{IR,CG,UT})}$ [59]	Zero-Shot	✗	✗	✗	✓
TA-SQL [53]	✗	✗	✗	Pandas-Like	✗
OpenSearch-SQL [67]	✓	✗	✗	SQL-Like	✓
RSL-SQL [4]	Zero-Shot	✗	✗	✗	✓
E-SQL [3]	✓	✗	✗	✗	✗
PET-SQL [31]	✓	✗	✗	✗	✓

2.4 Query Revision Module

This module refines candidate SQL queries to produce the final query. Its tasks involve correcting SQL syntax errors, refining the query based on execution results, and selecting the best query from the multiple candidates generated. We survey the representative strategies as follows and summarize the classification of existing works based on these strategies in Table 3.

LLM-Based. Similar to Self-Refine [40], generally, the process begins by presenting the candidate SQL query to LLMs alongside the original natural language query. In a zero-shot setting, the model is explicitly instructed to review the SQL query for potential issues, such as syntax errors, semantic misalignments, or missing components, and produce a corrected SQL query. DIN-SQL [49] employs this strategy effectively by integrating a self-correction module within its pipeline.

Execution-Guided. This strategy leverages the execution results of candidate SQL queries as a critical feedback mechanism to improve the accuracy. This process can be iterative, allowing multiple execution rounds, feedback analysis, and refinement until predefined termination conditions. MAC-SQL [60] employs a dedicated Refiner Agent that automates the error detection and correction process. Similarly, CHESS [59] starts with an initial draft query and executes it to gather feedback. The execution results, including error messages or outputs, are provided to the LLM, which adjusts and refines the SQL query accordingly.

Consistency-Based. Following the principle of self-consistency [62], this strategy generates multiple SQL queries through diverse reasoning paths, evaluates their execution results, and selects the most consistent query as the final output. C3-SQL [10] incorporates a Consistency Output module, where multiple SQL queries are executed and filtered, and a voting mechanism is applied to the execution results to identify the most consistent query. CHESS [59] selects the most consistent SQL query from three samples.

Unit-Test-Based. This strategy first generates multiple unit tests to highlight the differences between the candidate queries. Then, it selects the best query based on the evaluation results of the unit tests. To the best of our knowledge, $\text{CHESS}_{(\text{IR,CG,UT})}$ is the first approach that introduces this novel strategy.

Question Rewriting. This strategy aims to better guide LLMs in revising and constructing SQL queries by enriching or rewriting the natural language questions. For example, E-SQL [3] enriches the questions by incorporating relevant database items, candidate predicates, and SQL generation steps. DART-SQL [41] utilizes database content to clarify and disambiguate questions through rewriting.

Ranking and Selection. Given multiple candidate queries, this strategy ranks the candidate queries based on some criteria and then selects the top-1 query from the query pool. A representative approach employing this method is CHASE-SQL [48], which fine-tunes a model specialized for ranking and selecting queries.

Hybrid Approaches. Similar to the Candidate Generation module, many works combine multiple of the above strategies to achieve better results. We present the detailed classification in Table 3.

Horizontal Comparison. Self-refine is lightweight but prone to superficial edits and regressions. Execution-guided repair addresses syntax and missing objects but adds iterations and cannot resolve silent semantic mismatches. Consistency-based selection reduces randomness and outliers but consumes sampling budget and may amplify shared biases. Unit-test-based checks deliver strong disambiguation but require specification effort and are sensitive to test quality. Question rewriting clarifies constraints and schema cues but can introduce bias or leakage. Ranking and selection consolidate decisions but may misrank under shift and add scoring overhead.

Table 3: Classification of Query Revision strategies.

Approach	LLM Based	Execution Guided	Consistency Based	Unit Test	Rank Select	Question Rewrite
C3-SQL [10]	✗	Single-Turn	✓	✗	✗	✗
DIN-SQL [49]	✓	✗	✗	✗	✗	✗
MAC-SQL [60]	✗	Up to 3	✗	✗	✗	✗
$\text{CHESS}_{(\text{IR,SS,CG})}$ [59]	✗	Single-Turn	✓	✗	✗	✗
$\text{CHESS}_{(\text{IR,CG,UT})}$ [59]	✗	Single-Turn	✗	✓	✗	✗
OpenSearch-SQL [67]	✗	Single-Turn	✓	✗	✗	✗
RSL-SQL [4]	✗	Up to 5	✗	✗	✓	✗
E-SQL [3]	✗	✓	✗	✗	✗	✓
DART-SQL [41]	✗	✓	✗	✗	✗	✓
GSR [16]	✗	✓	✗	✗	✗	✗
PET-SQL [31]	✗	Single-Turn	✓	✗	✗	✗
CHASE-SQL [48]	✗	✓	✗	✗	✓	✗

3 NL2SQLBENCH: BENCHMARKING AND EVALUATION FRAMEWORK

We now introduce **NL2SQLBench**, our modular benchmarking and evaluation framework for LLM-enabled NL2SQL approaches. We first propose fine-grained evaluation metrics tailored to each module, then present our multi-agent benchmarking framework for NL2SQL approaches, thus facilitating a comprehensive experimental analysis and providing deeper insights into their effectiveness, efficiency, and potential areas for improvement.

3.1 Evaluation Metrics

We propose a set of metrics for each module, enabling a modularized and fine-grained evaluation of NL2SQL systems. By isolating the performance of individual modules, we facilitate a more nuanced understanding of their effectiveness and efficiency.

3.1.1 Metrics for Schema Selection. To evaluate the effectiveness of the Schema Selection module, we propose three metrics: *Precision*, *Recall*, and *F1-score*, for both table and column retrieval. These metrics provide a standardized way of assessing the module's ability to select relevant schema elements while minimizing irrelevant selections. Let R represent the set of relevant schema elements, and S represent the set of elements selected by the system. We define:

$$\text{Precision} : \mathrm{P} = |R \cap S| \,/\, |S| \tag{1}$$

$$\text{Recall} : \mathrm{R} = |R \cap S| \,/\, |R| \tag{2}$$

$$\text{F1-score} : \mathrm{F_1} = 2 \cdot P \times R \,/\, (P + R) \tag{3}$$

Precision measures the proportion of relevant elements retrieved, while Recall assesses the system's ability to identify all relevant schema elements. A high F1-score reflects a balance between both. These metrics are calculated separately at the table and column levels to evaluate schema selection performance.

3.1.2 Metrics for Candidate Generation. Evaluating the effectiveness of this module is critical, as its output directly influences downstream processes such as query revision. Existing benchmarking, such as Spider [71] and BIRD [28], proposed the *Execution Accuracy* (EX) metric as the proportion of examples for which the executed results of both the predicted and ground-truth SQLs are identical, relative to the overall number of test cases.

To provide a comprehensive evaluation, we propose more fine-grained metrics: *Correct Rate* (CR)[2], *Incorrect Rate* (IR), and *Error Rate* (ER), to evaluate a generated SQL query by assessing whether it produces (1) a Correct query (a correct result), (2) an Incorrect query (an incorrect result without runtime execution errors), or (3) an Error query (a runtime execution error), respectively.

Let C be the set of correct SQL queries, I be the set of Incorrect queries that execute without runtime errors, E be the set of queries that produce execution errors, and Q be the total number of queries generated. We define:

$$\text{Correct Rate} : \mathrm{CR} = |C| \,/\, |Q| \tag{4}$$

$$\text{Incorrect Rate} : \mathrm{IR} = |I| \,/\, |Q| \tag{5}$$

$$\text{Error Rate} : \mathrm{ER} = |E| \,/\, |Q| \tag{6}$$

These metrics satisfy $C \cup I \cup E = Q$ and C, I, E are pairwise disjoint, ensuring that each generated query falls into exactly one category and that CR + IR + ER = 1.

For SQL queries producing runtime execution errors, we classify them into the most common categories based on error messages and failure types such as *no such table/column*, *no such function*, *syntax error*, *timeout*, and *other error types*. This fine-grained categorization provides additional diagnostic insights into error analysis.

Furthermore, we adopt `Pass@k` for approaches producing multiple candidate queries, similar to CHESS [59]. Specifically, `Pass@k` measures the rate of producing at least one correct SQL query among `k` SQL queries produced per NL question. Formally,

$$\texttt{Pass@k} \;=\; \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\Big[\exists\, j \le k : \mathrm{Correct}\big(q_{i,j}\big) = 1\Big] \tag{7}$$

where the parameters are defined as follows.

- N: Total number of NL queries.
- k: Number of candidate SQL queries produced per NL query.
- $q_{i,j}$: The j^{th} candidate query generated for the i^{th} NL query.
- $\mathrm{Correct}(\cdot)$: Boolean function that returns 1 if a candidate query yields the correct execution result, and 0 otherwise.
- $\mathbb{I}(\cdot)$: Indicator function: 1 if the argument is true, 0 otherwise.

This metric is mainly to measure the upper limit of the achievable accuracy when producing multiple candidates, thus providing important references for evaluating the Candidate Generation module and the Query Revision module.

3.1.3 Metrics for Query Revision. To comprehensively evaluate the Query Revision module, we use the same primary metrics as for Candidate Generation—Correct Rate, Incorrect Rate, and Error Rate, with detailed error categorization. Additionally, we introduce new metrics—Correctness Improvement Rate (CI), Incorrect-to-Correct Rate (I2C), Error-to-Correct Rate (E2C), Correct-to-Incorrect Rate (C2I), and Correct-to-Error Rate (C2E)—to specifically measure this module's ability to enhance query quality and resolve errors without introducing regressions.

Let CR_{pre} and CR_{post} denote the Correct Rates before and after Query Revision. Similarly, let C_{pre} and C_{post} represent the Correct query sets, I_{pre} and I_{post} represent the Incorrect query sets, and E_{pre} and E_{post} represent the Error query sets before and after Query Revision, respectively. Then, we define:

$$\mathrm{CI} = \big(CR_{\text{post}} - CR_{\text{pre}}\big) \,/\, CR_{\text{pre}} \tag{8}$$

$$\mathrm{I2C} = |I_{\text{pre}} \cap C_{\text{post}}| \,/\, |I_{\text{pre}}| \tag{9}$$

$$\mathrm{E2C} = |E_{\text{pre}} \cap C_{\text{post}}| \,/\, |E_{\text{pre}}| \tag{10}$$

$$\mathrm{C2I} = |C_{\text{pre}} \cap I_{\text{post}}| \,/\, |C_{\text{pre}}| \tag{11}$$

$$\mathrm{C2E} = |C_{\text{pre}} \cap E_{\text{post}}| \,/\, |C_{\text{pre}}| \tag{12}$$

Among the new metrics, CI, I2C, and E2C capture improvements in query quality, while C2I and C2E identify potential regressions. Together with the primary metrics from the Candidate Generation module, these measures enable a comprehensive assessment of the module's strengths and weaknesses.

3.1.4 Efficiency Metrics. Although an increasing number of novel solutions have achieved impressive performance on NL2SQL leaderboards, most overlook computational cost and latency, which are critical for real-world deployment [34]. This oversight significantly limits their deployment in real-world environments with strict constraints on resources, budgets, and real-time responsiveness.

To bridge this critical gap, we propose explicitly measuring the efficiency of individual NL2SQL modules using two metrics: *#Tokens* (the total number of tokens for prompts and completions) and *#LLM Calls* (the number of LLM invocations)[3]. By systematically quantifying these metrics, our framework enables a detailed evaluation of computational overhead, fostering the development of NL2SQL solutions that are practically viable for widespread adoption.

3.2 Multi-Agent Benchmarking Framework

To facilitate the benchmarking, we develop a multi-agent benchmarking framework, as shown in the bottom half of Figure 1, comprising three specialized agents:

[2]The Correct Rate is equivalent to the Execution Accuracy used by Spider and BIRD.

[3]For the methods utilizing self-consistency on n outputs, we compute *#Tokens* and *#LLM Calls* by invoking the language model n times

Result Collecting Agent. This agent collects the results required for evaluation metrics from each module. It gathers selected schemas, candidate SQL queries, and refined SQL queries from the Schema Selection, Candidate Generation, and Query Revision modules, respectively. It also collects LLM usage statistics, such as token costs and the number of LLM calls. All the collected information is consolidated into structured JSON files. We provide an example of the collected JSON format in Appendix A.1 of our extended paper [19].

Postprocessing Agent. This agent transforms raw outputs from upstream modules into a standardized format for benchmarking. Its main functions include extracting gold schema from gold SQL queries as a reference for schema selection accuracy, executing both candidate and refined SQL to compare results with gold SQL for correctness and error identification, and serializing evaluation outputs into structured JSON files for further analysis. Additionally, the agent compiles performance statistics, reporting total executions, success and failure counts, and per-question summaries.

Benchmarking Agent. Utilizing the processed datasets, this agent performs a comprehensive and modular evaluation of each module. Specifically, it leverages the previously defined evaluation metrics to compute quantitative performance indicators for each module. For each module, the agent aggregates relevant statistics and analyzes these metrics across varying difficulty levels of questions, enabling a granular assessment. The resulting evaluations are recorded in a structured format, providing detailed performance breakdowns that support comparative analysis across different strategies.

NL2SQLBench is designed for broad adaptability across NL2SQL systems, LLMs, and evaluation datasets. The *Result Postprocessing* and *Benchmarking* agents function identically for all NL2SQL methods, requiring no method-specific changes. Only the *Result Collecting* agent adapts to each system, extracting and standardizing intermediate outputs with minimal wrapper code and configuration. **NL2SQLBench** allows users to switch LLMs or datasets via simple configuration without architectural modifications. Future updates will introduce an Orchestration agent to automate the evaluation workflow, enabling reproducible, end-to-end benchmarking of NL2SQL pipelines with minimal scripting effort.

4 EXPERIMENTS AND EVALUATION

We now present a comprehensive, modularized benchmarking of existing NL2SQL solutions, focusing on their three core modules, using our evaluation metrics and **NL2SQLBench** framework to systematically and thoroughly analyze their effectiveness and efficiency.

4.1 Experiment Settings

Datasets. To ensure a rigorous and comprehensive evaluation of NL2SQL approaches, we utilize the BIRD dataset [28], a large-scale and cross-domain dataset specifically designed to assess the capabilities of NL2SQL systems across diverse and complex scenarios, and the ScienceBenchmark dataset [75], which contains three real-world, highly domain-specific databases[4].

Evaluated Approaches. We choose a list of representative and open-sourced approaches[5] from the BIRD leaderboard: C3-SQL [10], DIN-SQL [49], MAC-SQL [60], TA-SQL [53], CHESS$_{(IR,SS,CG)}$ [59], CHESS$_{(IR,CG,UT)}$ [59], E-SQL [3], RSL-SQL [4], OpenSearch-SQL [67] and GSR [16]. These approaches cover diverse strategies for the three core modules mentioned previously.

Language Models. To ensure a fair and consistent comparison across all approaches while minimizing the experimental costs, we utilize DeepSeek-V3 [8] and GPT-4o-mini [46] as the unified LLMs for all evaluations[6]. For model-related hyperparameters such as temperature, maximum tokens, and prompts, we retain the original configurations used by each evaluated method to ensure objectivity in evaluation. For the approaches that require models for embedding and retrieval, we choose the model `bge-large-en-v1.5` [66].

4.2 Results for Schema Selection Module

Table 4: Analysis results of Schema Selection on BIRD using DeepSeek-V3. The best results are in bold and underlined while the second-best results are underlined.

Approach	Table Selection			Column Selection			Efficiency	
	Precision (%)	Recall (%)	F1-score (%)	Precision (%)	Recall (%)	F1-score (%)	#Tokens	#LLM calls
C3-SQL	50.03	98.80	64.63	27.88	95.03	41.76	15886	20
DIN-SQL	93.27	95.47	93.41	89.10	88.76	**88.01**	7360	**1**
MAC-SQL	32.74	**99.84**	46.70	15.84	**98.70**	26.12	**3179**	**1**
CHESS$_{(IR,SS,CG)}$	**94.50**	95.85	**94.35**	**89.78**	88.00	87.66	307894	78.56
TA-SQL	93.66	95.73	93.77	82.46	90.77	84.89	4249	**1**
RSL-SQL	88.53	97.09	91.21	55.03	93.01	63.37	5790	**1**
OpenSearch-SQL	32.80	99.62	46.72	16.97	97.35	27.49	6698	3

4.2.1 Overall performance. Table 4 reports the overall results of the schema selection module on the BIRD with DeepSeek-V3. The results of the remaining three settings are deferred to Appendix A.2 in our extended paper [19]. Across all four settings, we observe highly consistent rankings.

Table selection. As shown in Table 4, CHESS$_{(IR,SS,CG)}$ and TA-SQL achieved the top two F1 scores, followed closely by DIN-SQL and RSL-SQL. This pattern largely persists when swapping the LLM and switching to ScienceBenchmark, with minor shifts. These consistent outcomes show the effectiveness of their Multi-Stage Schema Pruning (as in CHESS$_{(IR,SS,CG)}$) and Preliminary-SQL Enhanced strategies (as in TA-SQL). Few-Shot CoT (DIN-SQL) also delivers strong, often third-best F1, suggesting that lightweight reasoning with carefully curated exemplars can narrow the gap to heavier multi-stage pipelines.

Column selection. On BIRD, DIN-SQL leads the column-level F1, with CHESS and TA-SQL close behind; On ScienceBenchmark, leadership alternates among CHESS and TA-SQL, with DIN-SQL remaining competitive. Compared to BIRD, ScienceBenchmark tends to depress column-selection precision, leading to lower column-F1 across systems. We conjecture this stems from the fact that the schema complexity of ScienceBenchmark datasets is higher than BIRD. Indeed, methods such as MAC-SQL and OpenSearch-SQL often attain very high recall at both table and column levels but suffer from low precision.

[4]The evaluation is conducted on the dev Set of BIRD and ScienceBenchmark, containing 1534 and 299 test cases respectively.

[5]Due to computational budget, we do not evaluate fine-tuned or RL-trained models.

[6]We use the DeepSeek-V3-0324 Release version.

***Insight 1**: Multi-Stage Schema Pruning, Pre-SQL Enhanced strategies, and Few-Shot CoT reasoning are consistently effective for both table and column selection. However, their gains depend on careful implementation details. The predominant issue is over-selection, especially on complex schemas (ScienceBenchmark). This suggests adding precision-oriented controls to curtail long tails of spurious columns.*

***Insight 2:** Relative rankings are stable across LLMs and datasets, with only mild re-ordering at the top. This robustness indicates the observed advantages stem from algorithmic choices rather than opportunistic synergy with a particular LLM or dataset.*

4.2.2 Cost efficiency and performance on large-scale databases. Across settings, we observe a cost-quality frontier. $\text{CHESS}_{(\text{IR,SS,CG})}$ attains top table/column selection but incurs substantial token cost and LLM calls pose significant barriers to deployment, where latency and cost matter. By contrast, TA-SQL and DIN-SQL deliver near-top F1 with a single call and low tokens, offering a favorable accuracy–efficiency trade-off.

These trends persist on ScienceBenchmark, which better approximates real-world schema density than academic suites such as Spider [71] and BIRD [28]. On ScienceBenchmark we consistently see lower column-selection precision and stronger over-selection pressure. From a scalability standpoint, schema selection cost often grows with the number of tables/columns included in prompts or retrieval stages. This underscores the need for future research into lightweight pruning strategies that maintain accuracy while minimizing computational overhead.

***Insight 3:** Multi-stage pipelines (e.g., CHESS) yield the highest accuracy but at high token/call budgets; single-call pre-SQL/CoT approaches (e.g., TA-SQL, DIN-SQL) provide the strongest production-ready trade-off. ScienceBenchmark stresses these dynamics with denser schemas, revealing precision-cost trade-off as a crucial concern for maintaining accuracy under strict cost/latency constraints.*

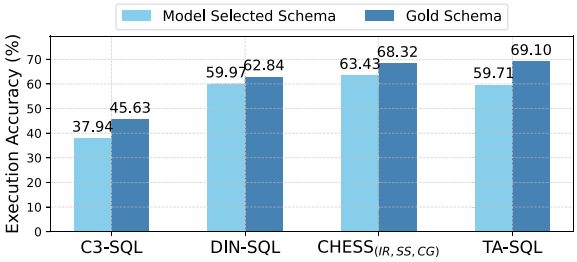


Figure 2: Execution accuracy using different schemas.

4.2.3 Influence of schema selection on overall performance. To assess the influence of accurate schema selection on the overall performance, we conducted experiments comparing execution accuracies using model-selected schemas versus gold schemas on some approaches[7] using BIRD dev set and Deepseek-V3. Figure 2 presents the execution accuracies for each approach under both conditions. The results indicate that the use of gold schemas significantly enhances execution accuracy across all evaluated approaches. Specifically, for C3-SQL, the execution accuracy increased from 37.94% to 45.63%, a significant improvement of 20.27% relatively. TA-SQL's execution accuracy improved from 59.71% to 69.10%, the highest increase among all approaches, with a gain of 15.73%. This observation confirms the critical role of accurate schema selection – the provision of the correct schemas to the subsequent modules eliminates errors arising from incorrect schema information, which is aligned with the observation of [12, 60]. While some recent studies [3, 38] suggest that schema selection is unnecessary with today's strongest LLMs, we contend it remains indispensable for robust NL2SQL systems. Not only are cutting-edge models often inaccessible under strict compute or budget constraints, but feeding entire enterprise schemas—potentially comprising thousands of tables and columns—can exceed model context windows, increase token costs, and ultimately degrade output quality[33].

[7]Due to budget limitations, we only chose C3-SQL, DIN-SQL, CHESS, and TA-SQL.

Table 5: Results of Candidate Generation on BIRD dev with two LLMs. The best results inside each block are bold+underlined, the second-best are underlined. We use `pass@1` for methods producing multiple candidates.

Approach	DeepSeek-V3					GPT-4o-mini				
	CR (%)	IR (%)	ER (%)	#Tok.	#Calls	CR (%)	IR (%)	ER (%)	#Tok.	#Calls
C3-SQL	38.14	50.07	11.80	11786	20	35.66	51.83	12.52	1353	20
DIN-SQL	59.58	35.52	4.89	6680	2	56.65	34.49	8.87	6523	2
MAC-SQL	58.74	33.83	7.43	2829	**1**	52.74	35.92	11.34	3031	**1**
$\text{CHESS}_{(\text{IR,SS,CG})}$	59.78	**33.12**	7.11	**1017**	**1**	53.42	41.03	**5.54**	**934**	**1**
$\text{CHESS}_{(\text{IR,CG,UT})}$	62.91	33.12	3.98	234394	20	55.08	32.01	12.91	228177	20
TA-SQL	59.71	34.42	5.87	2592	2	53.06	33.57	13.36	2427	2
GSR	57.37	37.09	5.48	3846	2	52.67	40.29	7.04	2006	2
E-SQL	57.50	35.07	7.43	13118	**1**	**58.02**	32.07	9.91	13031	**1**
RSL-SQL	59.00	33.38	7.62	11370	5	55.54	35.01	9.45	2219	5
OpenSearch-SQL	**66.10**	**30.83**	**3.06**	100095	21	56.52	35.33	8.15	8984	21

***Insight 4:** Accurate schema selection is critical in improving execution accuracy and preventing error propagation, and remains essential for robust real-world deployment due to practical constraints.*

4.2.4 Practical guide. Our analysis reveals that over-selection is prevalent, while Multi-Stage Schema Pruning achieves superior accuracy, it incurs substantial computational costs that may hinder real-world deployment. To address this trade-off, we recommend implementing a filter-and-refine pipeline that balances effectiveness and efficiency. In the first stage, employ lightweight retrieval methods to broadly filter irrelevant schemas, prioritizing high recall to minimize the risk of excluding relevant schemas. In the second stage, leverage LLMs with carefully designed prompts to perform precision-oriented refinement on the reduced schema space, focusing on eliminating irrelevant elements. Please refer to Appendix A.5 of the extended paper [19] for more details.

For resource-constrained environments, Preliminary-SQL Enhanced and Few-shot CoT strategies are favored, which leverage the inherent generation capabilities of LLMs without multiple calls, offering a practical balance between accuracy and efficiency.

4.3 Results for Candidate Generation Module

4.3.1 Overall performance. We evaluate `pass@1` for comparability. Table 5 and Table 6 show that, unlike schema selection, candidate generation is highly sensitive to both the dataset and the LLM—rankings shift, and there is no universal winner.

Table 6: Results of Candidate Generation on ScienceBenchmark. The best results are bold+underlined and the second-best are underlined. We report pass@1 for methods producing multiple candidates.

Approach	DeepSeek-V3					GPT-4o-mini				
	CR (%)	IR (%)	ER (%)	#Tok.	#Calls	CR (%)	IR (%)	ER (%)	#Tok.	#Calls
C3-SQL	54.18	42.81	3.01	12110	20	50.84	42.81	6.35	1350	20
DIN-SQL	52.84	44.82	2.34	6149	2	42.81	48.49	8.70	5932	2
MAC-SQL	47.16	42.81	10.03	3514	**1**	44.15	41.81	14.05	3734	**1**
CHESS$_{(IR,SS,CG)}$	37.92	55.37	6.71	978	**1**	**53.42**	**41.03**	**5.54**	934	**1**
CHESS$_{(IR,CG,UT)}$	54.85	39.46	5.69	259569	20	40.80	46.82	12.37	247969	20
TA-SQL	**59.53**	**38.46**	**2.01**	2712	2	50.17	43.81	6.02	2523	2
GSR	40.47	47.49	12.04	**824**	2	40.47	52.17	7.36	**766**	2
E-SQL	54.52	40.47	5.02	6542	**1**	38.80	45.82	15.38	12736	**1**
RSL-SQL	45.48	39.46	15.05	2877	5	40.47	52.84	6.69	1532	5
OpenSearch-SQL	57.19	39.13	3.68	103355	21	42.14	46.82	11.04	8355	21

BIRD. With DeepSeek-V3, OpenSearch-SQL attains the best Correct rate (CR) with the lowest Incorrect Rate (IR), and CHESS$_{(IR,CG,UT)}$ is a close second. With GPT-4o-mini, the ordering changes: E-SQL achieves the highest CR, and simpler pipelines such as DIN-SQL and TA-SQL remain competitive at low token budgets. These shifts indicate that candidate generation quality depends strongly on the LLM; the same pipeline can rank differently when the LLM changes.

ScienceBenchmark. With DeepSeek-V3, TA-SQL yields the strongest CR at low Error Rate (ER), while breadth-oriented systems like CHESS$_{(IR,CG,UT)}$ see degraded CR due to increased fragility on complex schemas. With GPT-4o-mini, CHESS$_{(IR,SS,CG)}$ becomes the most effective at pass@1 with *one* call and sub-1K tokens.

Error profile. Across all four blocks, the dominant failure mode is IR: syntactically valid but *semantically misaligned* SQL substantially outnumbers execution errors. Typical IR patterns include (i) wrong join path or missing bridge table, (ii) predicate misalignment, (iii) aggregation/GROUP BY/HAVING mismatches, and (iv) projection-level omissions or extra columns. ER is non-negligible for some complex databases on ScienceBenchmark, but even there IR constitutes the majority of errors; improving *semantic alignment* is therefore the primary lever for improving the accuracy.

Cost–efficiency. A clear cost–quality frontier emerges. Multi-candidate pipelines (e.g., OpenSearch-SQL, CHESS$_{(IR,CG,UT)}$) can lead on BIRD but often require tens of calls and very high token budgets; on ScienceBenchmark, their advantage weakens. In contrast, token-efficient pipelines (e.g., CHESS$_{(IR,SS,CG)}$, TA-SQL, sometimes DIN-SQL) achieve competitive CR at a fraction of the cost.

Insight 5: *The dominant failure mode is semantic mismatch. Adding semantic checks that align the SQL with the question will be helpful. Results shift between LLMs and datasets, indicating that performing evaluation on targeting databases and backbone LLMs before adopting specific NL2SQL approaches is necessary.*

We also conducted a detailed error analysis and result analysis on different difficulty levels of questions. Due to page limits, we present the full results in Appendix A.3 in our extended paper [19].

4.3.2 pass@k *evaluation for multiple candidates.* To evaluate methods that produce multiple candidate queries, we report pass@k for $k \in \{1, 5, 10, 15, 20\}$ across four settings: BIRD with DeepSeek-V3 and GPT-4o-mini, and ScienceBenchmark with DeepSeek-V3 and GPT-4o-mini. As illustrated in Figure 3, across all subfigures, the curves show the same shape: execution accuracy rises with larger k, confirming the benefit of generating multiple candidates. The marginal gains, however, diminish as k increases, indicating limited returns beyond a moderate number of candidates. The consistency among all subfigures indicates that our conclusion is robust to both the backbone LLM and the evaluation dataset. Since pass@k represents an upper bound on achievable accuracy, these results suggest that a well-designed Query Revision module—capable of reliably selecting the optimal query—has the potential to elevate the accuracy close to this theoretical limit.

Insight 6: *Increasing the number of candidates raises potential accuracy across all LLMs/datasets, but the gains diminish rapidly; if the Query Revision module can consistently select the best query from the candidates, overall accuracy can nearly reach the upper bound.*

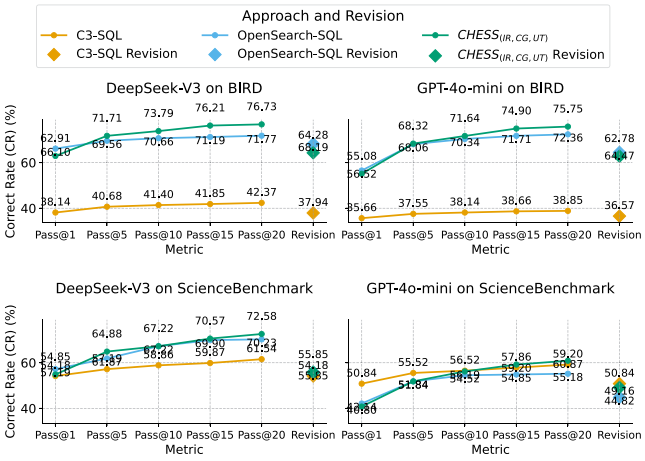


Figure 3: pass@k results for multiple-candidate approaches.

4.3.3 Practical guide. Our evaluation demonstrates that syntactically valid but semantically misaligned queries constitute the dominant failure case, far exceeding execution errors. This finding suggests that current candidate generation modules should consider semantic-based validation during the generation phase rather than deferring all validation to the revision stage. This early detection mechanism can prevent the propagation of semantic errors to downstream modules and reduce the burden on query revision.

For systems generating multiple candidates, our pass@k analysis reveals that increasing diversity ($k > 10$) yields diminishing returns; instead, practitioners should focus on improving the quality of top-ranked candidates through better strategies. In addition, using early-exit, and expanding to larger k only for high-uncertainty queries is a more cost-efficient approach.

4.4 Results for Query Revision Module

4.4.1 Overall performance. Figures 4 and 5 compare before vs. after the Revision stage on Correct Rate (CR), Incorrect Rate (IR), and Error Rate (ER) for all approaches.

BIRD. Across most systems, Revision raises CR while reducing IR and keeping ER low. The gains are largest for pipelines that already provide reasonably good candidates: Revision acts mainly

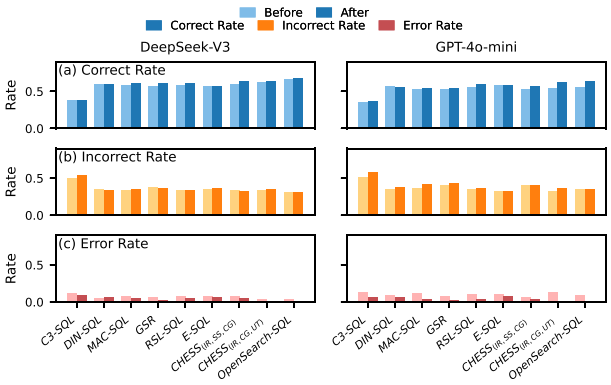


Figure 4: Analysis of the Query Revision module on BIRD.

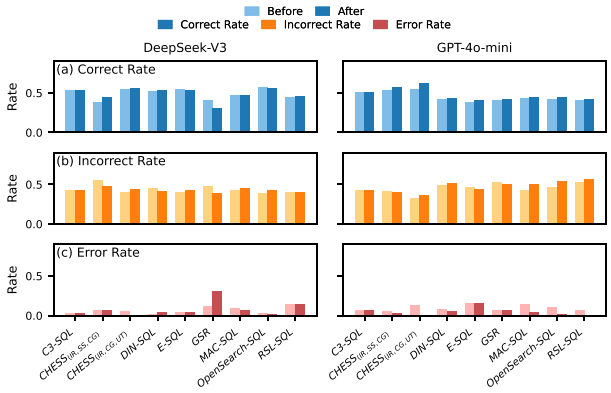
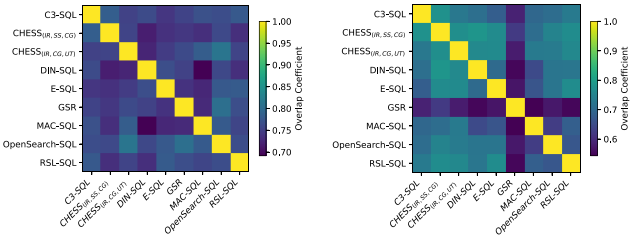


Figure 5: Analysis of the Query Revision module on ScienceBenchmark.

as a selector/repair step that fixes small semantic gaps and filters poor candidates. Methods that rank/select the best SQL query from multiple SQL candidates also improve, but their CR uplift depends on the revision policy's ability to reliably select the optimal query.

ScienceBenchmark. The improvements are more heterogeneous across systems. Some systems still obtain clear CR gains, but others see smaller gains or even a mild rise in IR. Furthermore, the extent of performance improvement varies depending on the backbone LLM employed, even when using the same NL2SQL approach. The divergence can be attributed to two compounding factors. First, the ScienceBenchmark dataset generally features more complex schemas and more challenging questions than the BIRD dataset, resulting in a more demanding evaluation of query revision strategies. Second, the query revision strategies of many NL2SQL approaches are, to some extent, specifically tailored to the BIRD dataset and therefore do not fully generalize to the ScienceBenchmark scenarios, struggling to address the unique challenges that the latter presents.

Error profile. In all settings, the predominant failure type after revision is still IR (syntactically valid yet semantically misaligned SQL), which greatly outnumbers ER. Therefore, the upper bound for further improvements in CR primarily depends on mitigating IR, while reducing ER is still necessary.



(a) **Overlap on the BIRD (using DeepSeek-V3)** (b) **Overlap on the ScienceBenchmark (using DeepSeek-V3)**

Figure 6: The coefficient heatmap of different solutions on Incorrect cases.

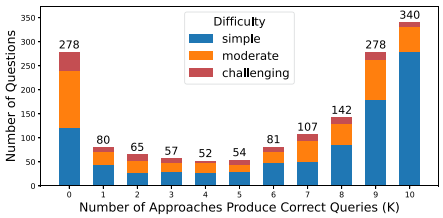


Figure 7: Breakdown of questions by number of approaches producing Correct queries (for different difficulty levels).

4.4.2 An in-depth analysis of incorrect cases. Through an in-depth analysis of incorrect cases, we observed an unexpectedly high degree of overlap in incorrect queries generated by different methods using DeepSeek-V3, as illustrated in Figure 6. Similar results using GPT-4o-mini are provided in Appendix A.4.2 of our extended paper [19]. Furthermore, as shown in Figure 7, a significant portion – 278 out of the total 1,534 questions from the BIRD Dev Set – could not be correctly solved by any of the evaluated approaches (the first vertical bar, which indicates no approach can produce correct SQL queries). This substantial overlap suggests a shared set of underlying limitations across existing NL2SQL methods, potentially involving insufficient semantic understanding, inadequate query generation strategies, or ineffective query revision mechanisms.

To uncover the root causes behind this, we conducted an additional human evaluation of these 278 problematic questions. Remarkably, the results revealed that a considerable fraction of these questions were paired with inaccurate gold SQL queries, which is aligned with the observation of [35]. This discovery highlights the often overlooked but critical issue of benchmark dataset quality and raises concerns that current assessments of NL2SQL methods on the BIRD dataset may have been compromised or misled by inaccuracies inherent in the benchmark. For a more detailed analysis of the BIRD dataset, please see Section 4.6.

Insight 7: *Our analysis shows significant overlap in incorrect queries across NL2SQL approaches, highlighting shared limitations. Many unresolved errors arise from inaccuracies in the BIRD dataset's gold SQL annotations. Future work should focus on improving benchmark annotation quality to enable more reliable evaluations and meaningful progress.*

Table 7: Effectiveness and efficiency on Query Revision on BIRD dev with DeepSeek-V3 vs GPT-4o-mini. We report `CI/I2C/E2C` and #Tokens, #LLM Calls. The best/second-best results are bold+underlined/underlined within each LLM.

Approach	DeepSeek-V3					GPT-4o-mini				
	CI (%)	I2C (%)	E2C (%)	#Tok.	#Calls	CI (%)	I2C (%)	E2C (%)	#Tok.	#Calls
C3-SQL	-0.51	1.43	4.97	–	–	2.56	2.52	8.33	–	–
DIN-SQL	0.66	2.02	21.33	4948	**1**	-0.11	4.16	19.12	4791	**1**
MAC-SQL	3.55	3.66	13.16	**1566**	2	4.45	1.45	21.84	1729	2
CHESS$_{(IR,SS,CG)}$	6.11	6.89	29.36	1844	**1**	6.72	10.49	35.29	**1683**	**1**
CHESS$_{(IR,CG,UT)}$	2.18	**15.16**	**47.54**	106126	21	14.04	16.09	38.38	115055	23.19
GSR	**7.27**	9.49	38.10	2220	2	2.60	4.85	21.3	2449	2
E-SQL	0.00	3.53	14.04	24924	2	1.91	7.52	28.29	25023	2
RSL-SQL	4.97	4.69	17.95	3315	2	9.15	10.06	35.86	2802	2
OpenSearch-SQL	3.16	8.67	34.04	6652	4.6	**14.07**	**17.71**	**51.20**	12969	6.64

Table 8: Effectiveness and efficiency on Query Revision on ScienceBenchmark dev with DeepSeek-V3 vs GPT-4o-mini. We report `CI/I2C/E2C` and #Tokens, #LLM Calls. The best/second-best results are bold+underlined/underlined.

Approach	DeepSeek-V3					GPT-4o-mini				
	CI (%)	I2C (%)	E2C (%)	#Tok.	#Calls	CI (%)	I2C (%)	E2C (%)	#Tok.	#Calls
C3-SQL	0	0	0	–	–	0	0	0	–	–
DIN-SQL	1.27	2.99	14.29	4349	**1**	2.34	1.38	15.38	4156	**1**
MAC-SQL	1.42	2.34	10	2395	2	0.76	3.20	9.52	2676	2
CHESS$_{(IR,SS,CG)}$	19.47	**14.55**	0	2319	**1**	**26.32**	**13.30**	14.29	1683	**1**
CHESS$_{(IR,CG,UT)}$	1.83	9.32	**52.94**	201372	23.42	20.49	12.14	**43.24**	115055	23.19
GSR	-24.8	14.08	33.33	**1757**	2	4.96	5.13	18.18	**1498**	2
E-SQL	-2.45	4.96	13.33	15088	2	5.17	6.57	4.35	21360	2
RSL-SQL	1.47	0.00	4.44	6616	2	5.79	0	35	3720	2
OpenSearch-SQL	-2.34	10.26	45.45	23239	8.26	6.35	7.14	33.33	34752	11.33

4.4.3 Detailed analysis on Query Revision performance. We evaluate the Query Revision module using Correct-Improvement (CI), Incorrect-to-Correct (I2C), and Error-to-Correct (E2C), along with efficiency (#tokens/#LLM calls). The consolidated results for BIRD and ScienceBenchmark are reported in Table 7 and Table 8, while the full details are in Appendix A.4.3 of our extended paper [19].

BIRD. Using DeepSeek-V3, methods that employ light, execution-guided strategies, notably GSR and CHESS$_{(IR,SS,CG)}$, obtain the highest CI with few calls and low tokens, while still posting healthy I2C/E2C. In contrast, unit-test-enabled pipelines such as CHESS$_{(IR,CG,UT)}$ achieve high I2C/E2C but only modest CI and at a very high cost. However, under GPT-4o-mini, the ranking shifts: breadth-oriented systems (OpenSearch-SQL, CHESS$_{(IR,CG,UT)}$) show the best CI on BIRD, indicating that GPT-4o-mini responds better to selection or unit-test style prompts than DeepSeek-V3. Compact pipelines (DIN-SQL, TA-SQL, CHESS$_{(IR,SS,CG)}$) remain cost-efficient with smaller but steady CI.

ScienceBenchmark. The results are polarized. CHESS$_{(IR,SS,CG)}$ delivers a large CI with a single call and low tokens. CHESS$_{(IR,CG,UT)}$ also posts high CI but at a very high budget. Several systems—even with decent I2C/E2C—show small or negative CI, meaning they flip correct queries to wrong during revision (i.e., high C2I risk).

Error Analysis. Across all four blocks, I2C/E2C improvements do not guarantee a high CI. When revision policies are aggressive, they may "fix" incorrect/error queries but also damage originally correct ones (C2I), netting a small or negative CI, as shown in the full detailed results in Appendix A.4.3 of our extended paper [19]. This explains the divergence we observe between good I2C/E2C figures and underwhelming CI on ScienceBenchmark/DeepSeek-V3 for several methods. Since our candidate generation analysis showed that IR (semantically misaligned yet executable SQL) dominates remaining errors, revision should prioritize semantic alignment and de-risk edits on already-correct SQL.

Insight 8: *Future exploration should focus on strategies not only to correct wrong answers, but also to avoid altering originally correct queries into incorrect ones. Methods that incorporate robust "do-no-harm" safeguards for originally correct SQL can be beneficial.*

Note that even the highest CI, achieved in the Query Revision section, is just merely 26.32%, which falls significantly short of a satisfactory threshold. Moreover, methods such as CHESS$_{(IR,CG,UT)}$ entail substantial token consumption and frequent LLM invocations, rendering them impractical for real-world deployment.

Insight 9: *All of the current Query Revision strategies exhibit limited effectiveness in improving the Correct Rate, underscoring a crucial need to develop methods that are both genuinely effective—achieving significant accuracy improvements—and practically efficient, minimizing cost and latency.*

Additionally, for approaches that produce multiple queries, as shown in Figure 3, while Correct Rates consistently improve with increasing `Pass@k`, the final Correct Rates after Query Revision remain notably lower, even below those at `Pass@5`. This indicates that the Query Revision module currently fails to approach the upper-bound effectiveness suggested by `Pass@5`.

Insight 10: *The gap between* `Pass@k` *upper bounds and post-revision results identifies Query Revision as a key bottleneck, highlighting the need for more effective strategies to fully capitalise on the diversity of multiple candidates.*

4.4.4 Practical guide. Our analysis reveals that current Query Revision strategies achieve limited effectiveness, with the predominant failure being the inability to correct queries that are syntactically valid but semantically misaligned. To address this critical bottleneck, we recommend a dual-strategy revision approach that combines execution-guided feedback with conservative validation. Detailed explanations are presented in Appendix A.5 of our extended paper [19]. For systems with multiple-candidate generation, query selection strategies are crucial. We recommend implementing ensemble-based selection that combines multiple signals: execution success, result plausibility, query complexity metrics, and LLM-based confidence scores. Finally, for real-world deployment, given the substantial token costs of iterative revision, practitioners should implement early stopping mechanisms based on execution feedback to avoid unnecessary costs.

4.5 End-to-End Overall Evaluation

We summarize the end-to-end overall evaluation of all approaches in Table 9 and Table 10. The cost per task is estimated based on the pricing guidelines of DeepSeek-V3[8] and GPT-4o-mini[9]. To simulate a realistic pricing scenario, we assume that half of the prompt tokens result in cache hits and the other half in cache misses.

[8] https://api-docs.deepseek.com/

[9] https://platform.openai.com/docs/pricing

Table 9: End-to-end results on BIRD dev with two LLMs. We report *Correct Rate*, efficiency (#Tokens, #LLM Calls) and *Cost*. The best/second-best results are bold+underlined/underlined within each LLM block.

Approach	DeepSeek-V3				GPT-4o-mini			
	Correct Rate (%)	#Tokens	#LLM Calls	Cost ($)	Correct Rate (%)	#Tokens	#LLM Calls	Cost ($)
C3-SQL	37.94	27673	40	0.0126	36.57	12332	40	0.0067
DIN-SQL	59.97	18988	**3**	0.0038	56.58	18511	**3**	0.0024
MAC-SQL	60.82	7574	4	0.0019	55.08	7867	4	0.0013
CHESS$_{(IR,SS,CG)}$	63.43	310756	80.6	0.0604	57.01	300061	80.5	0.0364
CHESS$_{(IR,CG,UT)}$	64.28	340601	41.8	0.0816	62.82	343232	43.2	0.0525
TA-SQL	59.71	6842	**3**	0.0017	53.06	6546	**3**	**0.0007**
GSR	61.54	**6066**	4	**0.0015**	54.04	**4455**	4	**0.0007**
E-SQL	57.50	38903	**3**	0.0074	59.13	38054	**3**	0.0047
RSL-SQL	61.93	20475	8	0.0041	60.63	10549	8	0.0013
OpenSearch-SQL	**68.19**	113446	28.6	0.0242	**64.47**	28569	10.7	0.0057

Table 10: End-to-end results on ScienceBenchmark dev with two LLMs. We report *Correct Rate*, efficiency (#Tokens, #LLM Calls) and *Cost*. The best/second-best results are bold+underlined/underlined within each LLM block.

Approach	DeepSeek-V3				GPT-4o-mini			
	Correct Rate (%)	#Tokens	#LLM Calls	Cost ($)	Correct Rate (%)	#Tokens	#LLM Calls	Cost ($)
C3-SQL	54.18	30979	40	0.0152	50.84	14179	40	0.0078
DIN-SQL	53.51	17299	**3**	0.0035	43.81	16660	**3**	0.0022
MAC-SQL	47.83	9290	4	0.0023	44.48	9577	4	0.0015
CHESS$_{(IR,SS,CG)}$	45.30	355983	92.59	0.0677	**57.01**	300061	80.5	0.0364
CHESS$_{(IR,CG,UT)}$	55.85	460941	43.42	0.1018	49.16	393208	47.6	0.0428
TA-SQL	**59.53**	7867	**3**	0.0018	50.17	7572	**3**	0.0008
GSR	30.43	**2581**	4	**0.0004**	42.47	**2264**	4	**0.0004**
E-SQL	53.18	21630	**3**	0.0044	40.80	34096	**3**	0.0042
RSL-SQL	46.15	10546	8	0.0021	42.81	11990	8	0.0018
OpenSearch-SQL	55.85	132884	32.26	0.0272	44.82	49371	15.34	0.0084

BIRD. With DeepSeek-V3, OpenSearch-SQL attains the highest CR (68%), about 29 calls and 113K tokens per query. CHESS variants are similarly accurate yet even more expensive. In contrast, compact pipelines—TA-SQL, MAC-SQL, GSR—reach 59–61% CR with 1–4 calls and 6–8K tokens, yielding much lower cost. The results obtained using GPT-4o-mini are highly similar, indicating the performance of the evaluated approaches is stable across LLMs.

ScienceBenchmark. With DeepSeek-V3, leadership shifts to TA-SQL (60% CR) using only 3 calls and sub-8K tokens. Breadth-heavy systems (CHESS$_{(IR,SS,CG)}$, CHESS$_{(IR,CG,UT)}$, OpenSearch-SQL) consume 30–90+ calls and 130K–460K+ tokens yet do not dominate CR. With GPT-4o-mini, CHESS$_{(IR,SS,CG)}$ achieves the best CR (57%) but at a very high cost (80 calls and 300K tokens). TA-SQL retains strong cost efficiency. The ranking shift indicates there is no universal winner on the ScienceBenchmark dataset: the best method depends on both the dataset and the backbone LLM.

Insight 11: *End-to-end rankings are LLM and dataset-dependent; validate on the target database(s) with the intended LLM before adoption is necessary.*

Cost-efficiency. While OpenSearch-SQL, CHESS$_{(IR,CG,UT)}$ and CHESS$_{(IR,SS,CG)}$ achieve superior accuracy on some workloads, they incur substantial token costs and LLM invocations. Recent studies, such as CHASE-SQL [48] and MCS-SQL [21], have also shown outstanding performance by increasing LLM calls to generate a large pool of candidate queries. However, we believe that while increasing the number of LLM calls can improve accuracy, this approach may lack adaptability in real-world deployment.

Insight 12: *While scaling LLM calls and token usage can enhance accuracy on some datasets, such strategies may undermine scalability and cost-effectiveness. Future work should aim to achieve high accuracy without proportional growth in token consumption.*

4.5.1 Practical guide. We recommend reporting CR@*budget* (e.g., CR at a fixed call or token cap) and selecting methods along the Pareto frontier instead of by CR alone. To achieve high accuracy without huge token consumption, we recommend implementing adaptive module selection that dynamically adjusts computational strategies based on query characteristics and system confidence signals. See Appendix A.5 of our extended paper [19] for full details.

4.6 New Discoveries on Existing Datasets

We conduct an in-depth analysis of BIRD and identified three key limitations that compromise its reliability as follows. Detailed examples are provided in Appendix A.6 of our extended paper [19].

4.6.1 Inaccurate Gold SQL Queries. As shown in Section 4.4, some NL queries in BIRD are paired with the wrong Gold SQL queries. Although [35] recently identified 106 incorrectly annotated queries in the BIRD dataset, their results still haven't exhaustively covered all annotation errors. The prevalence of such annotation errors raises concerns about the reliability and validity of existing datasets.

4.6.2 Strict Evaluation Rules. The BIRD benchmark currently assesses each NL question against a single gold SQL query, an assumption that proves overly restrictive in many realistic scenarios [12]. This over-strictness calls for more flexible and semantically aware evaluation rules—such as result-set equivalence or graded relevance metrics—to foster fairer assessments of future NL2SQL solutions.

4.6.3 Semantic Ambiguity. Existing datasets inevitably contain questions that exhibit a certain degree of semantic ambiguity [12]. Mitigating the effect of such ambiguity will require either (i) rewriting questions to eliminate ambiguity or (ii) devising evaluation rules that recognize ambiguity and reward semantically equivalent answers. Developing formal methods to define, detect, and quantify semantic ambiguity remains an open problem and constitutes a promising avenue for future research.

Insight 13: *Systematic auditing of gold annotations, adoption of evaluation rules that acknowledge multiple semantically equivalent queries, and explicit modeling of semantic ambiguity are crucial for reliable assessment and methodological progress. Tackling these issues, notably present in the BIRD dataset, will enable more accurate evaluations and advance robust NL2SQL solutions.*

4.6.4 Practical guide. We recommend developing semantically-aware evaluation frameworks that recognize and credit semantically equivalent but syntactically different SQL queries. Such improvements are vital for fair benchmarking and for training NL2SQL models that truly understand semantics rather than replicating annotation errors from noisy datasets. Detailed recommendations are provided in Appendix A.6 of our extended paper [19].

4.7 Leveraging NL2SQLBench for NL2SQL System Development

The modular architecture of **NL2SQLBench** enables practitioners to adopt a **compositional optimization** approach to NL2SQL development, analogous to assembling building blocks where each module can be independently diagnosed, optimized, and composed. To leverage this capability effectively, we recommend the following systematic workflow. First, establish a performance profile by running **NL2SQLBench** with different baselines on your target dataset to generate a comprehensive performance profile across all three core modules. This profile identifies the primary bottleneck. Second, apply targeted optimizations to the bottleneck module while monitoring cross-module impacts. Third, conduct a cost-benefit analysis at the module level using our effectiveness and efficiency metrics. Finally, leverage **NL2SQLBench** for A/B testing and continuous improvement. By treating NL2SQL as a modular optimization problem where each component can be independently analyzed, improved, and composed, rather than a monolithic system design challenge, practitioners can systematically navigate the accuracy-efficiency trade-off space and construct systems tailored to their specific deployment requirements. We provide a case study of leveraging **NL2SQLBench** in Appendix A.7 of our extended paper [19].

5 RELATED WORK

LLM-based NL2SQL. Since LLMs have demonstrated distinctive emergent abilities [63], LLM-based NL2SQL methods have become the most prominent solutions in the current NL2SQL landscape, as described in recent surveys [18, 34, 44, 57, 80]. Detailed discussions are presented in Appendix A.9 of our extended paper [19].

NL2SQL Datasets and Dataset Evaluation. Several datasets have been proposed recently to facilitate the development and evaluation of NL2SQL solutions. WikiSQL [79] and Spider [71] published several cross-domain datasets, which mainly contain lightweight databases and focus on database schema. BIRD [28] focuses on large databases and external knowledge, and has become a widely used NL2SQL dataset nowadays. ScienceBenchmark [75] is a highly domain-specific NL2SQL dataset in the field of scientific data analysis, which contains real-world complex queries and large databases. Recently proposed Spider-2.0 [22] provides a more realistic enterprise-level NL2SQL benchmark, encompassing multiple database systems, diverse SQL dialects, and numerous challenging tasks from real data engineering pipelines. In the line of related work that evaluates NL2SQL datasets, Mitsopoulou and Koutrika proposed a comprehensive analysis of existing NL2SQL datasets, and provided valuable insights into their capabilities and limitations, and how they affect training and evaluation of NL2SQL systems [43]. Liu et al. [35] further developed a benchmark for detecting and categorizing semantic errors in NL2SQL. Yang et al. [70] presents a method for detecting and fixing the errors in NL2SQL translation.

NL2SQL System Evaluations. Several experimental studies have been proposed to evaluate NL2SQL solutions. For example, Chang et al. [5] evaluated the robustness of different NL2SQL models. Pourreza and Rafiei [50] studied the limitations of existing evaluation metrics and conducted a benchmark through human evaluation and query rewriting. Gao et al. [14] undertook an assessment examining multiple prompting strategies and fine-tuning methods. Zhang et al. [73] evaluated the performance of different LLMs in sub-tasks of NL2SQL pipeline. Li et al. [24] proposed a testbed for evaluating NL2SQL systems from different perspectives.

Unlike previous efforts, our work, **NL2SQLBench**, is the first to introduce a set of fine-grained evaluation metrics for each module of the NL2SQL pipeline, develop a benchmarking framework, and conduct a comprehensive benchmarking of diverse strategies at the modular level. We summarize the differences between **NL2SQLBench** and existing NL2SQL evaluation frameworks in Table 11.

Table 11: Comparison of evaluation frameworks for NL2SQL datasets and systems.

Benchmarking Framework	Evaluation Objectives	E2E Eval	Modular Eval	Fine-grained Metrics	Error Analysis	Practical Guides
Text2sql Benchmarks [43]	Datasets	–	–	–	**Yes**	–
SQLDriller [70]	Datasets	–	–	–	**Yes**	–
NL2SQL-BUGs [35]	Datasets	–	–	–	**Yes**	–
DR-Spider [5]	Systems	**Yes**	No	No	**Yes**	Partial
Cross-Domain Text2sql [50]	Systems	**Yes**	No	No	**Yes**	Partial
Text2sql by LLMs [14]	Systems	**Yes**	No	No	No	Partial
Benchmarking Text2sql [73]	Systems	**Yes**	No	No	**Yes**	Partial
NL2SQL360 [24]	Systems	**Yes**	No	No	No	Partial
NL2SQLBench (ours)	Systems	**Yes**	**Yes**	**Yes**	**Yes**	**Detailed**

6 LIMITATIONS

Limited Scope of Evaluated Approaches. Our evaluation encompasses a selection of representative NL2SQL approaches from the BIRD leaderboard. However, due to code availability constraints and limited computational resources, not all existing NL2SQL methods were included. For instance, the studies [7, 15, 25, 27, 48, 52, 56] that fine-tune specialized models for specific tasks within the NL2SQL pipeline were excluded from our benchmarking. Future work should incorporate a broader array of approaches to enable a more comprehensive evaluation of the current landscape of NL2SQL systems.

Lack of Industry-Level Evaluation. Current benchmarks may not fully reflect the complexities of real-world industry workloads, including schema intricacy and query ambiguity. Future research should incorporate more realistic, industry-level evaluations to better capture practical deployment challenges.

7 CONCLUSIONS

We have systematically addressed the critical need for a unified, modular evaluation of LLM-enabled NL2SQL approaches. Specifically, we conducted a comprehensive review of the three core modules *Schema Selection*, *Candidate Generation*, and *Query Revision*. We proposed a novel set of fine-grained metrics and developed **NL2SQLBench**, a modular, multi-agent benchmarking framework. Our evaluation of representative NL2SQL approaches highlights substantial opportunities for improvement in both accuracy and computational efficiency Our in-depth analysis exposed critical shortcomings in current benchmark datasets and evaluation rules, underscoring the urgency for developing more accurate, robust, and standardized evaluation resources. By establishing this foundational benchmarking framework, our work aims to provide useful insights and practical guides for future NL2SQL development, ultimately benefiting both academic research and enterprise applications.

REFERENCES

[1] Anastasia Ailamaki, Samuel Madden, Daniel Abadi, Gustavo Alonso, Sihem Amer-Yahia, Magdalena Balazinska, Philip A Bernstein, Peter Boncz, Michael Cafarella, Surajit Chaudhuri, et al. 2025. The Cambridge Report on Database Research. *arXiv preprint arXiv:2504.11259* (2025).

[2] Arian Askari, Christian Poelitz, and Xinye Tang. 2025. Magic: Generating self-correction guideline for in-context text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 23433–23441.

[3] Hasan Alp Caferoğlu and Özgür Ulusoy. 2024. E-sql: Direct schema linking via question enrichment in text-to-sql. *arXiv preprint arXiv:2409.16751* (2024).

[4] Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, and Wei Chen. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. *arXiv preprint arXiv:2411.00073* (2024).

[5] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness. In *ICLR*.

[6] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2024. Navigate through Enigmatic Labyrinth A Survey of Chain of Thought Reasoning: Advances, Frontiers and Future. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 1173–1203. https://doi.org/10.18653/v1/2024.acl-long.65

[7] Team Cohere, Arash Ahmadian, Marwan Ahmed, Jay Alammar, Milad Alizadeh, Yazeed Alnumay, Sophia Althammer, Arkady Arkhangorodsky, Viraat Aryabumi, Dennis Aumiller, et al. 2025. Command A: An enterprise-ready large language model. arXiv:2504.00698 [cs.CL] https://arxiv.org/abs/2504.00698

[8] DeepSeek-AI. 2025. DeepSeek-V3-0324 Release. https://api-docs.deepseek.com/news/news250325. Accessed: October 31, 2025.

[9] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A Survey on In-context Learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1107–1128. https://doi.org/10.18653/v1/2024.emnlp-main.64

[10] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306 [cs.CL] https://arxiv.org/abs/2307.07306

[11] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining Small Language Models and Large Language Models for Zero-Shot NL2SQL. *Proc. VLDB Endow.* 17, 11 (July 2024), 2750–2763. https://doi.org/10.14778/3681954.3681960

[12] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... Not!

[13] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. 2021. Natural SQL: Making SQL Easier to Infer from Natural Language Specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Punta Cana, Dominican Republic, 2030–2042. https://doi.org/10.18653/v1/2021.findings-emnlp.174

[14] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (January 2024), 1132–1145. https://www.vldb.org/pvldb/vol17/p1132-gao.pdf

[15] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. *arXiv preprint arXiv:2411.08599* (2024). https://arxiv.org/abs/2411.08599

[16] GSR-SQL. 2025. LLM Prompting for Text2SQL via Gradual SQL Refinement. https://github.com/GSR-SQL/GSR. *GitHub repository*. Accessed: October 31, 2025.

[17] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4524–4535. https://doi.org/10.18653/v1/P19-1444

[18] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-Generation Database Interfaces: a Survey of LLM-based Text-to-SQL. *IEEE Transactions on Knowledge and Data Engineering* (2025), 1–20. https://doi.org/10.1109/TKDE.2025.3609486

[19] Shizheng Hou, Wenqi Pei, Nuo Chen, Quang-Trung Ta, Peng Lu, and Beng Chin Ooi. 2025. NL2SQLBench: A Modular Benchmarking Framework for LLM-Enabled NL2SQL Solutions (Extended Version). https://github.com/HOU-SZ/NL2SQLBench.

[20] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169* (2023).

[21] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (Eds.). Association for Computational Linguistics, Abu Dhabi, UAE, 337–353. https://aclanthology.org/2025.coling-main.24/

[22] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=XmProj9cPs

[23] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. 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)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 6943–6954. https://doi.org/10.18653/v1/2020.emnlp-main.564

[24] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *Proc. VLDB Endow.* 17, 11 (July 2024), 3318–3331. https://doi.org/10.14778/3681954.3682003

[25] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing High-Quality Text-to-SQL Data at Scale. *Proc. VLDB Endow.* 18, 11 (July 2025), 4695–4709. https://doi.org/10.14778/3749646.3749723

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

[27] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data* 2, 3, Article 127 (May 2024), 28 pages. https://doi.org/10.1145/3654930

[28] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already Serve as a Database Interface? A big Bench for Large-Scale Database Grounded Text-To-SQLs. *Advances in Neural Information Processing Systems* 36 (2023), 42330–42357.

[29] Xiuwen Li, Qifeng Cai, Yang Shu, Chenjuan Guo, and Bin Yang. 2025. AID-SQL: Adaptive In-Context Learning of Text-to-SQL with Difficulty-Aware Instruction and Retrieval-Augmented Generation . In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 3945–3957. https://doi.org/10.1109/ICDE65448.2025.00294

[30] Yinheng Li. 2023. A Practical Survey on Zero-Shot Prompt Design for In-Context Learning. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, Ruslan Mitkov and Galia Angelova (Eds.). INCOMA Ltd., Shoumen, Bulgaria, Varna, Bulgaria, 641–647. https://aclanthology.org/2023.ranlp-1.69/

[31] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. PET-SQL: A Prompt-enhanced Two-stage Text-to-SQL Framework with Cross-consistency.

[32] Jinqing Lian, Xinyi Liu, Yingxia Shao, Yang Dong, Ming Wang, Zhang Wei, Tianqi Wan, Ming Dong, and Hailin Yan. 2024. ChatBI: Towards Natural Language to Complex Business Intelligence SQL. *arXiv preprint arXiv:2405.00527* (2024).

[33] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. https://doi.org/10.1162/tacl_a_00638

[34] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025. A Survey of Text-to-SQL in the Era of LLMs: Where Are We, and Where Are We Going? *IEEE Trans. Knowl. Data Eng.* 37, 10 (2025), 5735–5754. https://doi.org/10.1109/TKDE.2025.3592032

[35] Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. 2025. NL2SQL-BUGs: A Benchmark for Detecting Semantic Errors in NL2SQL Translation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2* (Toronto ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 5662–5673. https://doi.org/10.1145/3711896.3737427

[36] Lin Long, Xijun Gu, Xinjie Sun, Wentao Ye, Haobo Wang, Sai Wu, Gang Chen, and Junbo Zhao. 2025. Bridging the Semantic Gap Between Text and Table: A

Case Study on NL2SQL. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=qmsX2R19p9

[37] Shuai Lyu, Haoran Luo, Zhonghong Ou, Yifan Zhu, Xiaoran Shang, Yang Qin, and Meina Song. 2025. SQL-o1: A Self-Reward Heuristic Dynamic Search Method for Text-to-SQL. *arXiv preprint arXiv:2502.11741* (2025).

[38] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models. In *NeurIPS 2024 Third Table Representation Learning Workshop*. https://openreview.net/forum?id=fglyh5pa7d

[39] Karime Maamari and Amine Mhedhbi. 2024. End-to-end text-to-sql generation within an analytics insight engine. *arXiv preprint arXiv:2406.12104* (2024).

[40] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2023), 46534–46594.

[41] Wenxin Mao, Ruiqi Wang, Jiyu Guo, Jichuan Zeng, Cuiyun Gao, Peiyi Han, and Chuanyi Liu. 2024. Enhancing Text-to-SQL Parsing through Question Rewriting and Execution-Guided Refinement. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 2009–2024. https://doi.org/10.18653/v1/2024.findings-acl.120

[42] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2025. Large language models: A survey. arXiv:2402.06196 [cs.CL] https://arxiv.org/abs/2402.06196

[43] Anna Mitsopoulou and Georgia Koutrika. 2025. Analysis of text-to-SQL benchmarks: limitations, challenges and opportunities. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025*. 199–212.

[44] Ali Mohammadjafari, Anthony S. Maida, and Raju Gottumukkala. 2024. From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems. arXiv:2410.01066 [cs.CL]

[45] Beng Chin Ooi, Shaofeng Cai, Gang Chen, Yanyan Shen, Kian-Lee Tan, Yuncheng Wu, Xiaokui Xiao, Naili Xing, Cong Yue, Lingze Zeng, Meihui Zhang, and Zhanhao Zhao. 2024. NeurDB: an AI-powered autonomous data system. *Sci. China Inf. Sci.* 67, 10 (2024). https://doi.org/10.1007/S11432-024-4125-9

[46] OpenAI. 2025. GPT-4o mini: Fast, affordable small model for focused tasks. https://platform.openai.com/docs/models/gpt-4o-mini Accessed: October 31, 2025.

[47] Wenqi Pei, Hailing Xu, Henry Hengyuan Zhao, CHEN Han, Zining Zhang, Shizheng Hou, Luo Pingyi, and Bingsheng He. 2025. Optimizing Small Language Models for NL2SQL. In *ICLR 2025 Third Workshop on Deep Learning for Code*. https://openreview.net/forum?id=xGkxWP2wE4

[48] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=CvGqMD5OtX

[49] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=p53QDxSIc5

[50] Mohammadreza Pourreza and Davood Rafiei. 2023. Evaluating Cross-Domain Text-to-SQL Models and Benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 1601–1611. https://doi.org/10.18653/v1/2023.emnlp-main.99

[51] Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 8212–8220. https://doi.org/10.18653/v1/2024.findings-emnlp.481

[52] Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan O Arik. 2025. Reasoning-SQL: Reinforcement Learning with SQL Tailored Partial Rewards for Reasoning-Enhanced Text-to-SQL. In *Second Conference on Language Modeling*. https://openreview.net/forum?id=HbwkIDWQgN

[53] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 5456–5471. https://doi.org/10.18653/v1/2024.findings-acl.324

[54] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: Making a Large Language Model a Better SQL Writer . In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 15–28. https://doi.org/10.1109/ICDE60146.2024.00009

[55] Jaydeep Sen, Fatma Ozcan, Abdul Quamar, Greg Stager, Ashish Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, and Karthik Sankaranarayanan. 2019. Natural Language Querying of Complex Business Intelligence Queries. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1997–2000. https://doi.org/10.1145/3299869.3320248

[56] Lei Sheng and Xu Shuai Shuai. 2025. CSC-SQL: Corrective Self-Consistency in Text-to-SQL via Reinforcement Learning. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (Eds.). The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Mumbai, India, 1473–1496. https://aclanthology.org/2025.findings-ijcnlp.91/

[57] Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2025. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *ACM Comput. Surv.* 58, 2, Article 54 (Sept. 2025), 37 pages. https://doi.org/10.1145/3737873

[58] Yewei Song, Saad Ezzini, Xunzhu Tang, Cedric Lothritz, Jacques Klein, Tegawendé Bissyandé, Andrey Boytsov, Ulrick Ble, and Anne Goujon. 2024. Enhancing Text-to-SQL translation for financial system design. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 252–262.

[59] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *arXiv preprint arXiv:2405.16755* (2024).

[60] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*. 540–557.

[61] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[62] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=1PL1NIMMrw

[63] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research* (2022). https://openreview.net/forum?id=yzkSU5zdwD Survey Certification.

[64] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=_VjQlMeSB_J

[65] Luoxuan Weng, Yinghao Tang, Yingchaojie Feng, Zhuo Chang, Ruiqin Chen, Haozhe Feng, Chen Hou, Danqing Huang, Yang Li, Huaming Rao, Haonan Wang, Canshi Wei, Xiaofeng Yang, Yuhui Zhang, Yifeng Zheng, Xiuqi Huang, Minfeng Zhu, Yuxin Ma, Bin Cui, Peng Chen, and Wei Chen. 2025. DataLab: A Unified Platform for LLM-Powered Business Intelligence. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. 4346–4359. https://doi.org/10.1109/ICDE65448.2025.00326

[66] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-Pack: Packed Resources For General Chinese Embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 641–649. https://doi.org/10.1145/3626772.3657878

[67] Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. *Proc. ACM Manag. Data* 3, 3, Article 194 (June 2025), 24 pages. https://doi.org/10.1145/3725331

[68] Yuanzhen Xie, Xinzhou Jin, Tao Xie, Matrixmxlin Matrixmxlin, Liang Chen, Chenyun Yu, Cheng Lei, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for Enhancing Attention: Improving LLM-based Text-to-SQL through Workflow Paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 10796–10816. https://doi.org/10.18653/v1/2024.findings-acl.641

[69] Siqiao Xue, Danrui Qi, Caigao Jiang, Fangyin Cheng, Keting Chen, Zhiping Zhang, Hongyang Zhang, Ganglin Wei, Wang Zhao, Fan Zhou, Hong Yi, Shaodong Liu,

Hongjun Yang, and Faqiang Chen. 2024. Demonstration of DB-GPT: Next Generation Data Interaction System Empowered by Large Language Models. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4365–4368. https://doi.org/10.14778/3685800.3685876

[70] Yicun Yang, Zhaoguo Wang, Yu Xia, Zhuoran Wei, Haoran Ding, Ruzica Piskac, Haibo Chen, and Jinyang Li. 2025. Automated Validating and Fixing of Text-to-SQL Translation with Execution Consistency. In *SIGMOD*.

[71] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425

[72] Hongwei Yuan, Xiu Tang, Ke Chen, Lidan Shou, Gang Chen, and Huan Li. 2025. CogSQL: A Cognitive Framework for Enhancing Large Language Models in Text-to-SQL Translation. In *AAAI*. 25778–25786. https://doi.org/10.1609/aaai.v39i24.34770

[73] Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the Text-to-SQL Capability of Large Language Models: A Comprehensive Evaluation. arXiv:2403.02951 [cs.CL] https://arxiv.org/abs/2403.02951

[74] Meihui Zhang, Zhaoxuan Ji, Zhaojing Luo, Yuncheng Wu, and Chengliang Chai. 2024. Applications and Challenges for Large Language Models: From Data Management Perspective. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 5530–5541. https://doi.org/10.1109/ICDE60146.2024.00441

[75] Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. ScienceBenchmark: A Complex Real-World Benchmark for Evaluating Natural Language to SQL Systems. *Proc. VLDB Endow.* 17, 4 (2023), 685–698.

[76] Fuheng Zhao, Shaleen Deep, Fotis Psallidas, Avrilia Floratou, Divyakant Agrawal, and Amr El Abbadi. 2025. Sphinteract: Resolving Ambiguities in NL2SQL through User Interaction. *Proc. VLDB Endow.* 18, 4 (2025), 1145–1158. https://doi.org/10.14778/3717755.3717772

[77] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A survey of large language models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

[78] Zhanhao Zhao, Shaofeng Cai, Haotian Gao, Hexiang Pan, Siqi Xiang, Naili Xing, Gang Chen, Beng Chin Ooi, Yanyan Shen, Yuncheng Wu, and Meihui Zhang. 2025. NeurDB: On the Design and Implementation of an AI-powered Autonomous Database. In *CIDR*.

[79] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR* (2017). arXiv:1709.00103

[80] Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. 2024. Large Language Model Enhanced Text-to-SQL Generation: A Survey. arXiv:2410.06011 [cs.DB]