# The Dawn of Natural Language to SQL: Are We Fully Ready?

Boyan Li
HKUST (GZ)
boyanli@hkust-gz.edu.cn

Yuyu Luo*
HKUST (GZ) / HKUST
yuyuluo@hkust-gz.edu.cn

Chengliang Chai
Beijing Institute of Technology
ccl@bit.edu.cn

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Nan Tang
HKUST (GZ) / HKUST
nantang@hkust-gz.edu.cn

## ABSTRACT

Translating users' natural language questions into SQL queries (*i.e.,* NL2SQL) significantly lowers the barriers to accessing relational databases. The emergence of Large Language Models has introduced a novel paradigm in NL2SQL tasks, enhancing capabilities dramatically. However, this raises a critical question: *Are we fully prepared to deploy NL2SQL models in production?*

To address the posed questions, we present a multi-angle NL2SQL evaluation framework, **NL2SQL360**, to facilitate the design and test of new NL2SQL methods for researchers. Through **NL2SQL360**, we conduct a detailed comparison of leading NL2SQL methods across a range of application scenarios, such as different data domains and SQL characteristics, offering valuable insights for selecting the most appropriate NL2SQL methods for specific needs. Moreover, we explore the NL2SQL design space, leveraging **NL2SQL360** to automate the identification of an optimal NL2SQL solution tailored to user-specific needs. Specifically, **NL2SQL360** identifies an effective NL2SQL method, **SuperSQL**, distinguished under the Spider dataset using the execution accuracy metric. Remarkably, **SuperSQL** achieves competitive performance with execution accuracy of **87%** and **62.66%** on the Spider and BIRD test sets, respectively.

## 1 INTRODUCTION

Natural Language to SQL (NL2SQL), which converts a natural language query (NL) into an SQL query (SQL), can significantly lower the barrier for both lay users and expert users in accessing massive datasets and deriving insights [6, 11, 13, 23, 29–35, 38, 46, 47, 53, 56].
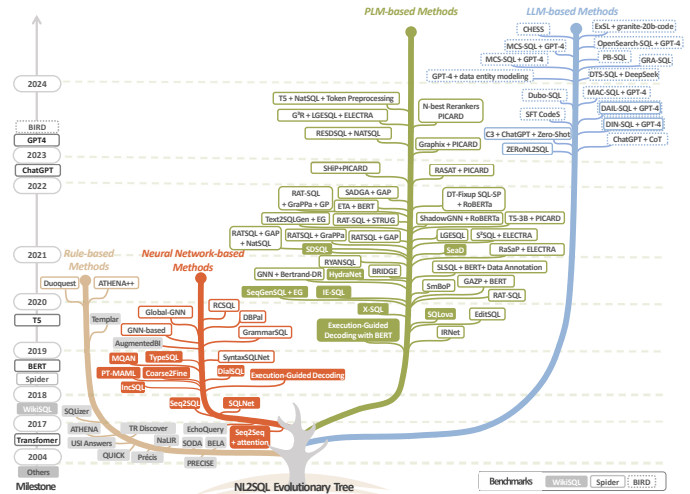
**Figure 1: An Overview of NL2SQL Methods.**
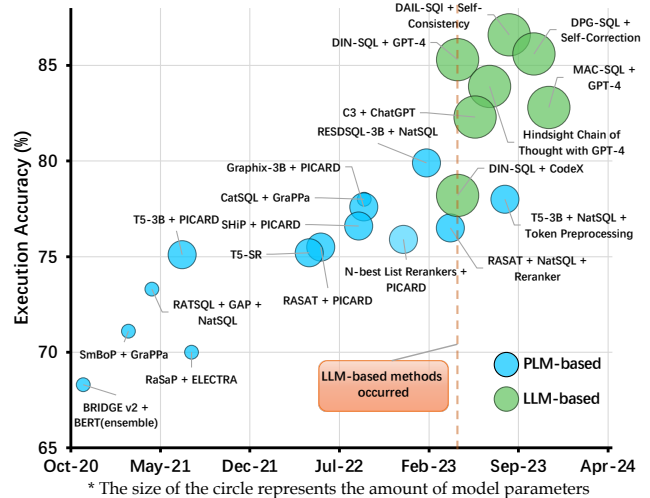


**Figure 2: Evolution of PLM- and LLM-based NL2SQL Models.**

Especially being empowered by the recent advances of large language models, the performance of NL2SQL solutions has been significantly improved. The trend of providing NL2SQL solutions by database vendors has shifted from a myth to a must-go.

Despite all these efforts in tackling NL2SQL, there are still many important questions, from where we are now, what NL2SQL research topic should be studied next for researchers, to which method one should apply to a specific application for practitioners – this paper systematically examines and answers these questions.

| (a) Test Results in Domain of *Competition* | | |
|---|---|---|
| 1 SFT CodeS-15B | 85.5% | ● |
| 2 RESDSQL-3B + NatSQL | 83.9% | ◆ |
| 3 RESDSQL-3B | 82.3% | ◆ |
| 4 Graphix + PICARD | 80.6% | ◆ |
| 4 DAIL-SQL | 80.6% | ○ |
| 5 DAIL-SQL (SC) | 79.0% | ○ |
| 6 DIN-SQL | 75.8% | ○ |
| 7 C3SQL | 69.4% | ○ |

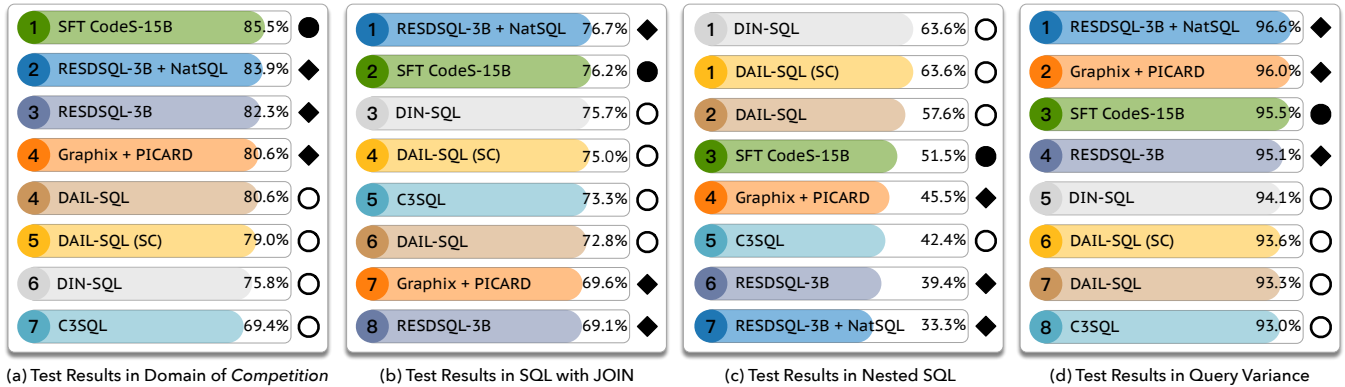| (b) Test Results in SQL with JOIN | | |
|---|---|---|
| 1 RESDSQL-3B + NatSQL | 76.7% | ◆ |
| 2 SFT CodeS-15B | 76.2% | ● |
| 3 DIN-SQL | 75.7% | ○ |
| 4 DAIL-SQL (SC) | 75.0% | ○ |
| 5 C3SQL | 73.3% | ○ |
| 6 DAIL-SQL | 72.8% | ○ |
| 7 Graphix + PICARD | 69.6% | ◆ |
| 8 RESDSQL-3B | 69.1% | ◆ |

| (c) Test Results in Nested SQL | | |
|---|---|---|
| 1 DIN-SQL | 63.6% | ○ |
| 1 DAIL-SQL (SC) | 63.6% | ○ |
| 2 DAIL-SQL | 57.6% | ○ |
| 3 SFT CodeS-15B | 51.5% | ● |
| 4 Graphix + PICARD | 45.5% | ◆ |
| 5 C3SQL | 42.4% | ○ |
| 6 RESDSQL-3B | 39.4% | ◆ |
| 7 RESDSQL-3B + NatSQL | 33.3% | ◆ |

| (d) Test Results in Query Variance | | |
|---|---|---|
| 1 RESDSQL-3B + NatSQL | 96.6% | ◆ |
| 2 Graphix + PICARD | 96.0% | ◆ |
| 3 SFT CodeS-15B | 95.5% | ● |
| 4 RESDSQL-3B | 95.1% | ◆ |
| 5 DIN-SQL | 94.1% | ○ |
| 6 DAIL-SQL (SC) | 93.6% | ○ |
| 7 DAIL-SQL | 93.3% | ○ |
| 8 C3SQL | 93.0% | ○ |

**Figure 3: NL2SQL Models on Spider from Different Angles (○: Prompting LLM, ●: Fine-tuning LLM, ◆: Fine-tuning PLM).**

**Q1: Where Are We Now?** Figure 1 shows the evolution of NL2SQL methods in the last two decades, from rule-based methods, deep neural network-based methods, tunable pre-trained language models (PLMs), to giant large language models (LLMs), alongside the development of benchmarks like Spider [58] and BIRD [26]. Note that LLMs (*e.g.*, GPT-4 [37] and Llama2 [50]) are larger language models compared to PLMs (e.g., GPT-2 [40] and BART [21]) and exhibit advanced language understanding and emergent abilities [36, 48, 57, 62]. Employing PLMs for the NL2SQL task requires fine-tuning on task-specific datasets, while harnessing LLMs for this task can be done through prompts (in-context learning) for all kinds of LLMs or fine-tuning (*i.e.,* instruction following) for open-source LLMs only [60].

Figure 2 compares the accuracy of PLM-based (blue dots) and LLM-based (green dots) NL2SQL models on Spider leaderboard [58]. It shows that LLM-based NL2SQL models started in Feb 2023 (DIN-SQL + CodeX) with comparable accuracy to PLM-based models. However, with the fast evolution of LLMs, the performance gap between LLM- and PLM-based models has been widening, highlighting the advantages of LLM-based approaches.

**Q2: Are LLM-based Models the Clear Winner?** Based on Figure 2, *can we conclude that LLM-based models are "the choice" for any NL2SQL application?* In other words, is selecting the model ranked at the top of the leaderboard always the best strategy?

Correctly answering this question is crucial in helping *researchers* and *practitioners* design and select the right model for different needs. Let's consider classical Business Intelligence (BI) use cases.

[*Various Data Domains.*] BI platforms like Tableau [1] often have various database domains (*e.g.,* movies and sports) with unique schemas and terminologies. An ideal NL2SQL model must generalize across these varied domains while adapting to each specific domain to meet ad-hoc requirements effectively.

[*Complex SQL operations.*] Real-world applications often require the execution of complex SQL queries, involving advanced operations such as multiple JOINs, nested queries, and aggregation functions. The capability to accurately generate complex queries is an important criterion for evaluating NL2SQL models.

[*New Linguistic Phenomena.*] For the same query intent, different users may pose NL questions with different abbreviations, synonyms, and question styles. Thus, the ability of an NL2SQL model to accurately interpret various NL query variants is crucial.

EXAMPLE 1. *Figure 3 compares the SOTA PLM- and LLM-based models from different angles on the Spider development dataset in terms of the Execution-Accuracy metric.*

[*Various Data Domains*] *Figure 3(a) compares different models in the Competition domain. The result shows that fine-tuning-based LLM/PLM methods outperform all prompt-based LLM methods. Specifically, the best PLM-based method, RESDSQL-3B+NatSQL [23], achieves 83.9% execution accuracy, which outperforms the best prompt-based LLM method, DAILSQL (with GPT-4) [11], by 3.3%. The above observations suggest that fine-tuning is a crucial strategy for enhancing the domain adaptation capabilities of NL2SQL models.*

[*Complex SQL operations*] *Figure 3(b) compares different models on use cases with only SQL queries with JOIN operators. It shows that the PLM-based method RESDSQL-3B+NatSQL [23] is ranked at the top, outperforming all LLM-based methods.*

*However, when we compare different methods on use cases with only nested SQL queries, as shown in Figure 3(c), we observe that the LLM-based methods generally outperform PLM-based methods.*

[*New Linguistic Phenomena*] *We also compute the average accuracy of the methods on different linguistic phenomena (e.g., "Return all customers whose total consumption is greater than 1000" vs. "What is the list of customers who spent more than 1,000?"). Figure 3(d) shows that although both types of methods perform well, fine-tuned LLM and PLM for NL2SQL are superior to prompting LLM for NL2SQL. This is primarily because fine-tuned models better align different query variants with database schemas.* □

Example 1 shows that *one size does not fit all*; that is, no NL2SQL model is a clear winner on different usage scenarios, even powered by currently the most powerful LLM GPT-4. In fact, real-world scenarios are much more complicated than what can be examined in public NL2SQL benchmarks such as Spider and BIRD. *Therefore, there is an urgent need for tools that can help systematically evaluate NL2SQL models from different angles on a given benchmark.*

**Q3: Can we combine the best of both worlds and design a super NL2SQL model?** The question following **Q1** and **Q2** is: if there is no single winner in different scenarios, can we design a super NL2SQL model that combines the merits of both PLMs and LLMs and is robust for different scenarios.

**Contributions.** In this paper, we present **NL2SQL360**, a testbed designed to systematically evaluate PLM- and LLM-based NL2SQL

models across different benchmarks from multiple perspectives. NL2SQL360 can help researchers and practitioners better evaluate NL2SQL models on their specific scenarios, uncover insightful experimental findings, and design a superior NL2SQL model that is more robust than SOTA solutions. Our contributions are as follows.

**(1) NL2SQL360: multi-angle NL2SQL evaluation.** We design a testbed, NL2SQL360, for fine-grained evaluation of NL2SQL solutions. NL2SQL360 can assess different NL2SQL methods against established benchmarks or tailor their evaluations based on specific criteria (*e.g.,* varying data domains or SQL characteristics). (Section 3)

**(2) New experimental findings.** We tested 13 LLM-based and 7 PLM-based NL2SQL solutions on the Spider and BIRD datasets, varying in 15 different settings to analyze their performance in various usage scenarios (Section 4). The key findings are as follows:

*(i) Accuracy.* Fine-tuning is crucial for enhancing performance. Specifically, LLM-based methods with fine-tuning excel in the EX metric, while PLM-based methods lead in the EM metric. Furthermore, PLM/LLM-based methods can be distinguished as winners in subsets of SQL with specific characteristics. For example, methods using GPT-4 perform notably better with subqueries.

*(ii) NL Query Variance.* For generating the same target SQL from different NL Queries, LLMs and PLMs fine-tuned on scenariospecific data exhibit stronger stability.

*(iii) Domain Adaption.* For NL2SQL tasks across different domains, there is no clear winner between LLM-based and PLM-based methods. However, in-domain data during the fine-tuning process is crucial for model performance in specific domains.

*(iv) The Impact of Corpus in Pre-training.* Our experiments reveal that after fine-tuning, LLMs pre-trained on code-specific datasets—like CodeLlama-7B, StarCoder-7B, and Deepseek-Coder-7B—outperform Llama2-7B, which is trained on general text, in NL2SQL tasks. This highlights the significant impact of an LLM's pre-training data domain, or its intrinsic code capabilities, on its performance in specialized tasks such as NL2SQL.

**(3) SuperSQL: A robust NL2SQL model.** We systematically categorize and analyze the most representative NL2SQL modules based on LLMs and PLMs, highlighting their commonalities and distinct features. Building on this exploration, we propose **SuperSQL**, which achieves competitive execution accuracy of **87%** on the Spider test set and **62.66%** on the BIRD test set. (Section 5)

**(4) What needs to be done next.** Based on our experimental findings, design space exploration, and the implementation and testing of **SuperSQL**, we identify three future research opportunities: i) enhancing the trustworthiness of NL2SQL methods, which includes handling ambiguous NL queries, diagnosing the match between the NL query and the predicted SQL, and interpreting the query results back to the NL query; ii) developing cost-effective NL2SQL solutions; and iii) automatically and adaptively generating training data (NL, SQL) based on evaluation results. (Section 6)

## 2 NATURAL LANGUAGE TO SQL

Let $\mathcal{N}$ be an NL query, $\mathcal{D}$ be a relational database with $n$ tables $\{T_1, \ldots, T_n\}$. The problem of natural language to SQL (NL2SQL) is to generate an SQL query $Q$ based on $\mathcal{N}$ and the database $\mathcal{D}$.

Next, we describe related work by categorizing recent LLM-based/PLM-based NL2SQL solutions into a taxonomy. We close this section by discussing the limitations of the existing works.

### 2.1 Related Works: A Bird's-Eye View

Figure 1 illustrates an evolutionary tree of NL2SQL techniques, categorized into four main branches: rule-based methods, neural network-based methods, PLM-based, and LLM-based methods.

**Rule-based Methods.** Early work relied on pre-defined rules or semantic parsers [17, 18, 22, 43]. For example, NaLIR [22] uses a syntactic parser and handcrafted rules to convert NL queries into SQL queries. However, these methods are limited in adaptability, scalability, and generalization.

**Neural Network-based Methods.** To address these limitations, researchers began using neural networks to translate NL queries to SQL queries. Several large-scale benchmark datasets such as WikiSQL [61] and Spider [58] were released. Sequence-to-sequence NL2SQL methods such as IRNet [15] were developed. IRNet encodes NL queries and database schemas with an encoder and generates SQL queries with a decoder.

**PLM-based Methods.** Around 2017, the introduction of the Transformer [51] and the Spider dataset led to the rise of neural network-based methods, which soon dominated the field. Models like BERT [8] and T5 [41] heralded the era of pre-trained language models, achieving top results on benchmark datasets [23, 25, 45]. For instance, RESDSQL [23], a top performer on the Spider leaderboard, uses a two-stage framework: it first identifies relevant schema elements from the NL query, then constructs the SQL query.

**LLM-based Methods.** The emergence of large language models like ChatGPT and GPT-4 [37] has revolutionized NL2SQL solutions. LLM-based methods now dominate the NL2SQL landscape [9, 11, 24, 38, 52]. For instance, DAIL-SQL [11] uses GPT-4 with prompt engineering, achieving competitive results on the Spider dataset.

Given the growth trend observed in the NL2SQL evolutionary tree, we anticipate that LLM-based/PLM-based NL2SQL methods will continue to dominate the field in the coming years. Therefore, it is important for us to fully understand the capabilities, limitations, and potential improvements of these NL2SQL methods.

**Key Modules in NL2SQL Systems.** Table 1 categorizes state-of-the-art NL2SQL methods based on backbone models and several key components. Roughly speaking, recent competitive methods adopt language models as the backbone for NL2SQL translation, either using giant and API-based large language models such as GPT-4 or tunable language models like T5 and Llama.

The schema linking module is vital to the NL2SQL process, as it enhances the accuracy and relevance of generated SQL queries by incorporating database content. This underscores the importance of understanding both the schema and content of databases for NL2SQL tasks. During SQL generation, PLM-based methods use beam search-like strategies (*e.g.,* PICARD [45]) to find optimal tokens within SQL syntax constraints, while LLM-based methods use greedy strategies. For post-processing, LLM-based methods employ heuristic prompting strategies like Self-Correction and Self-Consistency to refine outputs to better match intended SQL queries.

Table 1: Taxonomy of PLM- and LLM-based NL2SQL Methods.

| Types | | Methods | Backbone Models | Example Selection (Few-shot) | Schema Linking | DB Content | SQL Generation Strategy | | | Post-processing Strategy |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | Multi-Step | Intermediate Representation | Decoding Strategy | |
| LLM-based | Prompting | DIN-SQL [38] | GPT-4 | Manual | ✓ | ✗ | Classification Decomposition | NatSQL | Greedy Search | Self-Correction |
| | | DAIL-SQL [11] (with Self-Consistency) | GPT-4 | Similarity-based | ✗ | ✗ | ✗ | ✗ | Greedy Search | Self-Consistency |
| | | MAC-SQL [52] | GPT-4 | N/A | ✓ | ✗ | Sub-question Decomposer | ✗ | Greedy Search | Refiner |
| | | C3-SQL [9] | GPT-3.5 | N/A | ✓ | ✗ | ✗ | ✗ | Greedy Search | Self-Consistency |
| | | CodeS [24] | StarCoder | Similarity-based | ✓ | ✓ | ✗ | ✗ | Beam Search | Execution-Guided SQL Selector |
| | Fine-tuning | SFT CodeS [24] | StarCoder | N/A | ✓ | ✓ | ✗ | ✗ | Beam Search | Execution-Guided SQL Selector |
| PLM-based | Fine-tuning | RESDSQL + NatSQL [23] | T5 | N/A | ✓ | ✓ | Skeleton Parsing | NatSQL | Beam Search | Execution-Guided SQL Selector |
| | | Graphix + PICARD [25] | T5 | N/A | ✓ | ✓ | ✗ | ✗ | PICARD | ✗ |
| | | N-best Rerankers + PICARD [59] | T5 | N/A | ✓ | ✓ | ✗ | ✗ | PICARD | N-best Rerankers |
| | | T5 + NatSQL + Token Preprocessing [42] | T5 | N/A | ✓ | ✓ | ✗ | NatSQL | Greedy Search | ✗ |
| | | RASAT + PICARD [39] | T5 | N/A | ✓ | ✓ | ✗ | ✗ | PICARD | ✗ |
| | | SHiP + PICARD [16] | T5 | N/A | ✗ | ✓ | ✗ | ✗ | PICARD | ✗ |
| | | T5 + PICARD [45] | T5 | N/A | ✗ | ✓ | ✗ | ✗ | PICARD | ✗ |
| | | RATSQL + GAP + NatSQL [10] | BART | N/A | ✓ | ✓ | ✗ | NatSQL | ✗ | ✗ |
| | | BRIDGE v2 [28] | BERT | N/A | ✗ | ✓ | ✗ | ✗ | Schema-Consistency Guided Decoding | ✗ |

## 2.2 Existing Experiments and Their Limitations

**Existing Experiments.** There are several experimental studies relevant to our research. For example, Gao et al. [11] evaluated the potential of open-source LLMs for NL2SQL tasks through prompt engineering. Rajkumar et al. [43] explored the capabilities of the Codex language model in handling the NL2SQL task under zero-shot and few-shot settings. Gkini et al. [12] conducted an in-depth evaluation of parsing-based and keyword-based NL2SQL. While the first two studies mainly focused on evaluating LLM-based NL2SQL solutions, the third investigated parsing-based NL2SQL methods.

**Their Limitations.** Existing experiments have several limitations.

(1) *Overlook the Usage Scenarios.* Existing evaluations typically report overall results on the entire benchmark datasets (*e.g.,* Spider). While this provides a broad overview, it falls short in offering detailed comparisons across specific subsets of the data (see Figure 3). For example, we can filter the evaluated datasets based on distinct SQL characteristics or database domains, which could yield valuable insights into the relative effectiveness of different NL2SQL models for particular SQL query types or domain-specific scenarios.

(2) *Lack of Direct and Comprehensive Comparisons.* One primary limitation is that many recent NL2SQL solutions, especially those based on LLM and PLM, have not been systematically compared on well-established benchmarks and customized datasets.

(3) *Limited Exploration of the NL2SQL Design Space.* Current NL2SQL research and practice have limited exploration of the NL2SQL framework's design space using both LLM and PLM-based modules. This restricts our understanding of how different architectural and functional modules from both LLM and PLM can be synergistically incorporated to enhance NL2SQL solutions.

## 3 NL2SQL360: A TESTBED FOR NL2SQL

Figure 4 shows the framework of our testbed, **NL2SQL360**, comprising six core components, as discussed below.
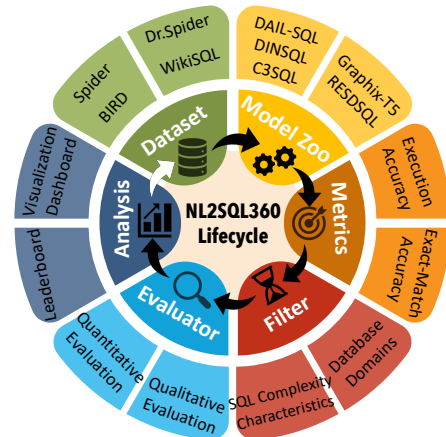


Figure 4: An Overview of **NL2SQL360**.

**Benchmark Datasts.** This module maintains widely-used benchmarks: Spider [58], BIRD [26], Spider-Realistic [7], Dr.Spider [4], KaggleDBQA [19], WikiSQL [61], etc.

**Model Zoo.** This module hosts a collection of competitive and open-source NL2SQL models featured on the Spider and BIRD leaderboards. It mainly includes LLM-based and PLM-based methods.

**Dataset Filter.** Traditional evaluations, which average performance across whole benchmark datasets, miss nuanced NL2SQL insights for different scenarios (Section 1). To address this, we select specific benchmark subsets, including particular databases, NL, and SQL queries, highlighting unique traits like query complexity, database schema diversity, and SQL features such as JOIN operations or nested queries.

Therefore, we introduce a dataset filtering mechanism in our **NL2SQL360**. This allows for the segregation of testing datasets into more focused subsets based on various criteria:

(1) **Scenario-1: SQL Complexity**. This scenario differentiates SQL queries by complexity, from straightforward to intricate queries

with multiple clauses and conditions. The classification follows the criteria established by Spider [58], aiming to evaluate how well NL2SQL methods handle varying levels of SQL difficulty.

(2) **Scenario-2: SQL Characteristics**. It examines SQL queries that primarily utilize specific features, such as JOIN operations, subqueries, or aggregate functions. By categorizing queries based on these characteristics, we can evaluate an NL2SQL system's ability to manage distinct SQL functionalities. *For example, business intelligence platforms often handle analytic queries with nested subqueries.*

(3) **Scenario-3: Data Domains**. This scenario explores the system's performance across various data domains, such as finance, healthcare, and retail. By categorizing NL2SQL databases according to their data domains, we provide a structured framework for evaluating domain-specific capabilities and potential limitations.

(4) **Scenario-4: Query Variance Testing**. It assesses the NL2SQL system's robustness and flexibility in handling variations in natural language queries. It tests the NL2SQL system's response to different phrasings and structures, measuring user-friendliness and adaptability to diverse linguistic styles. We use a variety of natural language queries from NL2SQL datasets as testing samples.

**Evaluation Metrics.** We support a set of widely-accepted metrics. Specifically, we adopt *Execution Accuracy* (EX) and *Exact Match Accuracy* (EM) [58] to assess the effectiveness of the generated SQL queries. In addition, we use the *Valid Efficiency Score* (VES) [26] to measure the efficiency of generating valid SQL queries.

To further evaluate the robustness and flexibility of NL2SQL solutions in handling variations in natural language queries, we propose a new metric called *Query Variance Testing*. This metric assesses how well the models can adapt to different forms of NL queries.

Given a SQL query $Q_i$, there typically exist multiple corresponding NL queries, denoted as pairs $\{(N_1, Q_i), (N_2, Q_i), \ldots, (N_m, Q_i)\}$. In evaluating an NL2SQL model, these NL and SQL query pairs are incorporated into the test set only if the model accurately processes at least one pair among them. This allows us to construct a specific test set for each model to compute their average accuracy.

The formula for computing *QVT* accuracy is defined as follows:

$$QVT = \frac{1}{M} \sum_{i=1}^{M} \left( \frac{\sum_{j=1}^{m_i} \mathbb{1}\left(\mathcal{F}(N_{ij}) = Q_i\right)}{m_i} \right) \quad (1)$$

where:

- $M$ is the total number of SQL queries in the test set.
- $m_i$ is the number of natural language query variations corresponding to the SQL query $Q_i$.
- $\mathcal{F}(N_{ij})$ represents the SQL query generated by the NL2SQL model for the $j$-th natural language query variation of $Q_i$.
- $\mathbb{1}(\cdot)$ is the indicator function that returns 1 if the query results inside are equal, and 0 otherwise.

**Executor and Logs.** Users can tailor the evaluation workflow of NL2SQL models, setting parameters like hyper-parameters and metrics. The testbed then automatically runs these models on benchmarks (e.g., Spider) and custom subsets (e.g., nested queries), logging every outcome. These logs offer detailed insights into each model's performance, serving as the resource for model analysis.

**Table 2: Spider *vs.* BIRD Dataset Statistics.**

| Dataset | #-Tables / DB | | | #-Columns / DB | | | #-Columns / Tables | | | #-PKs / DB | | | #-FKs / DB | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min | Max | Avg | Min | Max | Avg | Min | Max | Avg | Min | Max | Avg | Min | Max | Avg |
| Spider Train Set | 2 | 26 | 5.4 | 6 | 352 | 27.8 | 2 | 48 | 5.1 | 0 | 18 | 4.8 | 0 | 25 | 5.0 |
| BIRD Train Set | 2 | 65 | 7.6 | 6 | 455 | 51.3 | 1 | 62 | 6.8 | 0 | 65 | 6.7 | 0 | 61 | 6.1 |
| Spider Dev Set | 2 | 11 | 4.1 | 7 | 56 | 22.1 | 2 | 32 | 5.4 | 1 | 10 | 3.7 | 1 | 11 | 3.2 |
| BIRD Dev Set | 3 | 13 | 6.8 | 11 | 199 | 72.5 | 2 | 115 | 10.6 | 2 | 13 | 6.5 | 1 | 29 | 9.3 |

**Evaluator.** Leveraging data from Logs, the Evaluator automatically generates quantitative assessments, presented in easily interpretable formats like tables or leaderboards. Additionally, our testbed offers visualization tools and a dashboard for interactive analysis, allowing users to compare NL2SQL solutions across dimensions such as database domains and SQL characteristics.

# 4 EXPERIMENTS

## 4.1 Experimental Settings

**Datasets.** We use the development sets of Spider [58] and BIRD [26] as our experimental datasets, which contain 1034 and 1534 (NL, SQL) samples, respectively. The SQL structure from the BIRD dataset is more complex and includes some keywords not covered by Spider, such as CASE, IIF, etc. This added complexity challenges the model's NL2SQL ability. In addition, the databases in BIRD are more complex than those in Spider, as shown in Table 2.

**Methods.** We evaluate the state-of-the-art open-source LLM-based and PLM-based NL2SQL methods.

*Prompt-based LLMs.* We compare 4 prompt-based methods:

(1) DINSQL [38] decomposes the generation of SQL queries into different sub-problems and designs different prompts for each sub-problem to instruct GPT-4 to generate final SQL queries.

(2) DAILSQL [11] encodes the question and database schema in SQL code style. It selects few-shot examples based on their structural (skeleton) similarities and query similarities. These elements are combined into an efficient prompt to guide GPT-4.

(3) DAILSQL(SC) [11] is the version of DAILSQL with a Self-Consistency (SC) strategy for post-processing.

(4) C3SQL [9] uses schema linking filtering and a tailored calibration bias prompt with GPT-3.5 for SQL query generation, incorporating a self-consistency strategy for post-processing.

*Fine-tuning-based LLMs.* We evaluate 9 fine-tuning-based methods.

(5-8) SFT CodeS (1B/3B/7B/15B) [24]: CodeS is incrementally pre-trained based on StarCoder [27] using a large SQL-related corpus, which has demonstrated outstanding performance on many challenging NL2SQL benchmarks. In the following experiments, we use SFT CodeS which is fine-tuned with Spider or BIRD datasets. There are 4 versions of SFT CodeS family models in our experiments.

(9) Llama2-7B [50] uses an optimized Transformer as an auto-regressive language model, pre-trained on a vast corpus by Meta.

(10) Llama3-8B [2] on over 15T tokens of data – a training dataset 7x larger than that used for Llama 2, including 4x more code.

(11) StarCoder-7B [27] is a Code LLM that has been trained on permissively licensed data from GitHub. The data encompasses a

wide range of content, including code from over 80 programming languages, Git commits, GitHub issues, and Jupyter notebooks.

(12) CodeLlama-7B [44] is an enhanced variant of Llama2, refined with additional training on code repository datasets.

(13) Deepseek-Coder-7B [14] is trained on project-level code corpora and fill-in-the-blank tasks to boost code completion.

*PLM-based NL2SQL.* We evaluate 7 the state-of-the-art methods:

(1) Graphix-3B+PICARD [25] integrates a pre-trained T5-3B transformer with graph-aware enhancements for NL2SQL tasks, utilizing PICARD [45] to enhance performance.

(2-4) RESDSQL(Base/Large/3B) [23] introduces a ranking-enhanced encoding and skeleton-aware decoding to separate schema linking from skeleton parsing.

(5-7) RESDSQL(Base/Large/3B)+NatSQL [23] is the version incorporated with NatSQL [10] for better performance. There are 6 versions of RESDSQL family models used in the experiments.

**Metrics.** We evaluate different methods on Exact Match Accuracy (EM), Execution Accuracy (EX), Query Variance Testing (QVT), Valid Efficiency Socre (VES), Token Efficiency, and Latency metrics.

**Hardware and Platform.** All experiments are conducted on an Ubuntu 22.04.3 LTS server equipped with 512GB RAM and two 40-core Intel(R) Xeon(R) Platinum 8383C CPUs @ 2.70GHz. For the supervised fine-tuning of LLM experiments, we use 8 NVIDIA A800 (80GB) GPUs to fine-tune the models.

## 4.2 Experiments on Evaluating Accuracy

**Exp-1: Accuracy vs. SQL Complexity.** We evaluated all methods by varying SQL complexity using Spider and BIRD development sets. We computed Execution Accuracy (**EX**) and Exact Match Accuracy (**EM**) metrics. Note that we retrained RESDSQL from scratch on the BIRD train set but did not include NatSQL due to unavailable code. Additionally, We omitted DINSQL due to GPT resource constraints.

Table 3 and Table 4 report the results. The state-of-the-art (SOTA) **EX** and **EM** in specific SQL complexity are marked as **orange** and **blue** in the table, respectively.

*Insights based on the* **EX** *metric.* As shown in Table 3 and Table 4, we find that the EX of the LLM-based method exceeded the PLM-based method in different difficulty subsets. Particularly, in Table 4, DAILSQL(SC) outperforms LLM-based SOTA method SFT CodeS-15B on the Challenging subset, which may benefit from GPT-4's stronger reasoning capabilities.

*Insights based on the* **EM** *metric.* In Table 3, we find that LLM-based methods after supervised fine-tuning generally have higher EM performance than prompt-based LLM methods. After fine-tuning, both the LLM- and PLM-based model's output aligns more closely with the specific dataset's data distribution, leading it to predict SQL structures similar to those in that dataset.

> **Finding 1.** *Fine-tuning is an essential strategy to improve performance. Specifically, LLM-based methods with fine-tuning achieve the best overall results on the EX metric, while PLM-based methods perform best on the EM metric overall.*

**Table 3: Accuracy *vs.* SQL Complexity in Spider-Dev.**

| Types | | Methods | Metrics | Spider-Dev | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | Easy | Med. | Hard | Extra | All |
| LLM-based | Prompting | C3SQL | EX | 92.7 | 85.2 | 77.6 | 62.0 | 82.0 |
| | | | EM | 80.2 | 43.5 | 35.6 | 18.1 | 46.9 |
| | | DINSQL | EX | 92.3 | 87.4 | 76.4 | 62.7 | 82.8 |
| | | | EM | 82.7 | 65.5 | 42.0 | 30.7 | 60.1 |
| | | DAILSQL | EX | 91.5 | 89.2 | 77.0 | 60.2 | 83.1 |
| | | | EM | 89.5 | 74.2 | 55.5 | 45.2 | 70.0 |
| | | DAILSQL(SC) | EX | 91.5 | 90.1 | 75.3 | 62.7 | 83.6 |
| | | | EM | 88.3 | 73.5 | 54.0 | 41.6 | 68.7 |
| | Fine-tuning | SFT CodeS-1B | EX | 92.3 | 83.6 | 70.1 | 49.4 | 77.9 |
| | | | EM | 91.5 | 74.4 | 65.5 | 41.0 | 71.7 |
| | | SFT CodeS-3B | EX | 94.8 | 88.3 | 75.3 | 60.8 | 83.3 |
| | | | EM | **94.4** | 80.7 | 67.8 | 49.4 | 76.8 |
| | | SFT CodeS-7B | EX | 94.8 | **91.0** | 75.3 | **66.9** | **85.4** |
| | | | EM | 92.7 | **85.2** | 67.8 | 56.0 | 79.4 |
| | | SFT CodeS-15B | EX | **95.6** | 90.4 | **78.2** | 61.4 | 84.9 |
| | | | EM | 93.1 | 83.4 | 67.2 | 54.2 | 78.3 |
| PLM-based | | RESDSQL-3B | EX | 94.8 | 87.7 | 73.0 | 56.0 | 81.8 |
| | | | EM | 94.0 | 83.0 | 66.7 | 53.0 | 78.0 |
| | | RESDSQL-3B + NatSQL | EX | 94.4 | 87.9 | 77.0 | 66.3 | 84.1 |
| | | | EM | 93.1 | 83.0 | **70.1** | **65.7** | **80.5** |
| | | Graphix-3B + PICARD | EX | 92.3 | 86.3 | 73.6 | 57.2 | 80.9 |
| | | | EM | 91.9 | 82.3 | 65.5 | 53.0 | 77.1 |
| Hybrid | | **SuperSQL** | EX | 94.4 | **91.3** (0.3 ↑) | **83.3** (5.1 ↑) | **68.7** (1.8 ↑) | **87.0** (1.6 ↑) |
| | | | EM | 90.3 | 76.7 | 61.5 | 44.0 | 72.1 |

**Table 4: Accuracy *vs.* SQL Complexity in BIRD-Dev.**

| Types | | Methods | Metrics | BIRD-Dev | | | |
|---|---|---|---|---|---|---|---|
| | | | | Simple | Moderate | Challenging | All |
| LLM-based | Prompting | C3SQL | EX | 58.9 | 38.5 | 31.9 | 50.2 |
| | | DAILSQL | EX | 62.5 | 43.2 | 37.5 | 54.3 |
| | | DAILSQL(SC) | EX | 63.0 | 45.6 | **43.1** | 55.9 |
| | Fine-tuning | SFT CodeS-1B | EX | 58.7 | 37.6 | 36.8 | 50.3 |
| | | SFT CodeS-3B | EX | 62.8 | 44.3 | 38.2 | 54.9 |
| | | SFT CodeS-7B | EX | 64.6 | 46.9 | 40.3 | 57.0 |
| | | SFT CodeS-15B | EX | **65.8** | **48.8** | 42.4 | **58.5** |
| PLM-based | | RESDSQL-Base | EX | 42.3 | 20.2 | 16.0 | 33.1 |
| | | RESDSQL-Large | EX | 46.5 | 27.7 | 22.9 | 38.6 |
| | | RESDSQL-3B | EX | 53.5 | 33.3 | 16.7 | 43.9 |
| Hybrid | | **SuperSQL** | EX | **66.9** (1.1↑) | 46.5 | **43.8** (0.7↑) | **58.5** |

**Exp-2: Accuracy vs. SQL Characteristics.** Real-world applications often require generating SQL queries involving advanced operations like subqueries, logical connectors, ORDER BY, and multiple JOINs. Therefore, we will evaluate the capability of NL2SQL models to accurately generate SQL queries with varying characteristics.

To this end, we classify SQL queries based on four criteria: (1) the presence of subqueries, (2) the number of logical connectors, (3) the use of ORDER BY, and (4) the number of JOINs. Note that our **NL2SQL360** supports SQL query filtering based on individual SQL clauses, their combinations, or user-defined conditions. However, due to space constraints, we demonstrate only four representative aspects. We run all methods on these four subsets of SQL queries and compute their EX metrics.

We classify LLM-based methods into prompt-based and fine-tuning-based. Figure 5 shows the EX performance distribution across subsets of the Spider and BIRD datasets. Figure 6 and 7 provide detailed results. The bar chart presents the overall EX for each method, with methods on the x-axis and subsets on the y-axis.
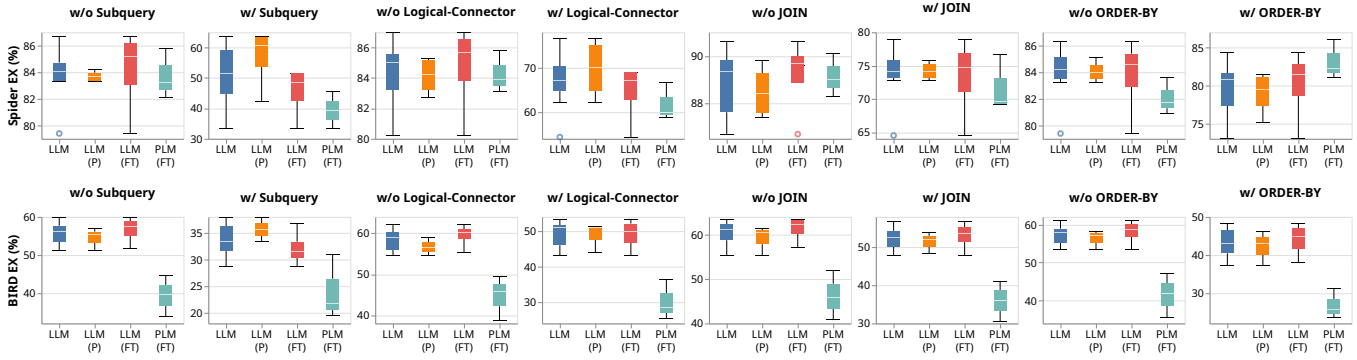
Figure 5: EX *vs.* SQL Characteristics. (LLM (P): Prompt-based LLMs, LLM (FT): Fine-tuned LLMs, PLM (FT): Fine-tuned PLMs.)



Figure 6: EX *vs.* SQL Characteristics on Spider.



Figure 7: EX *vs.* SQL Characteristics on BIRD.

**Exp-2.1: #-Subquery.** As shown in Figure 6 and Figure 7, all methods perform worst in cases with subqueries, indicating that reasoning through subqueries is a challenging task. Figure 5 shows that in scenarios without subqueries, the LLM-based methods slightly outperform the PLM-based methods on Spider and significantly outperform them on BIRD on average. In scenarios with subqueries, the LLM-based methods excel on both datasets.

This is because generating SQL with subqueries requires the model to first consider the subquery and then generate the entire SQL, demanding strong reasoning abilities. We find that all LLM-based methods, especially those prompted by GPT-4, perform better in subquery, surpassing both fine-tuned LLM-based methods and PLM-based methods. This suggests that the model's inherent reasoning ability is crucial for processing SQL with subqueries.

Finding 2. *In scenarios involving subqueries, LLM-based methods outperform PLM-based methods overall, with methods using GPT-4 (i.e., prompt-based LLM) showing particularly better performance. The inherent reasoning ability of these models is likely crucial for success in predicting the subqueries.*

**Exp-2.2: #-Logical Connector.** Logical Connectors (*e.g.,* AND, OR) are used to link conditions, filter query results, and perform other operations, making it essential to understand the model's performance with respect to logical connectors.

Without Logical Connectors (Figure 5), LLM-based methods do not outperform PLM-based methods on the Spider dataset. However, on the more complex BIRD dataset (Table 2), LLM-based methods excel. When Logical Connectors are needed, LLM-based methods consistently outperform PLM-based methods on both datasets.

Finding 3. *In scenarios where Logical Connectors are required, the LLM-based methods are better than the PLM-based methods.*

**Exp-2.3: #-JOIN.** In many usage scenarios, we need to generate SQL queries with JOINs across multiple tables. This challenges the model's ability to correctly understand complex database schemas.

*SQL without JOIN.* As shown in Figure 5, in scenarios without JOIN operations, LLM-based and PLM-based methods show inconsistent performance on Spider and BIRD, with no clear winner. Figure 6 and Figure 7 provide similar insights.

*SQL with JOIN.* However, for scenarios requiring JOIN operations, LLM-based methods outperform PLM-based methods in both datasets. This could be due to the JOIN operation's need for understanding complex database schemas, where LLMs typically excel due to their superior context-understanding capabilities.
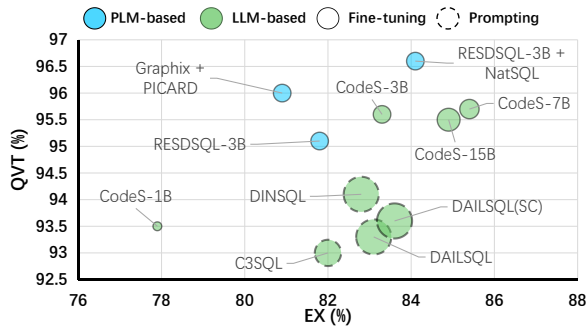
Figure 8: QVT *vs.* Execution Accuracy (EX).

*Impact of NatSQL.* In Figure 6, for SQL queries with JOIN, DINSQL works best in prompt-based methods, while RESDSQL-3B+NatSQL is the best among PLM-based methods. Both utilize NatSQL [10] as an intermediate representation, likely benefiting from its streamlined form that omits JOIN keywords and reduces schema item prediction, thus easing SQL prediction in JOIN scenarios.

> **Finding 4.** *In scenarios involving* JOIN *operations, LLM-based methods outperform PLM-based methods. Taking NatSQL as an intermediate representation reduces the complexity of predicting JOIN operations and potentially enhances the model performance.*

**Exp-2.4: #-ORDER BY.** As shown in Figure 5, we observed that without the ORDER BY clause, LLM-based methods outperform PLM-based methods on both the Spider and BIRD datasets. However, with the ORDER BY clause, LLM-based methods underperform compared to PLM-based methods on the Spider dataset, while they outperform PLM-based methods on the BIRD dataset. This difference might be because the BIRD dataset is more complex than the Spider dataset.

> **Finding 5.** *In scenarios with the* ORDER BY *clause, the performance of PLM/LLM-based methods varies across different datasets. Generally, LLM-based methods demonstrate stronger generalization capability.*

**Exp-3: Query Variance Testing.** We evaluate the NL2SQL system's adaptability to diverse natural language phrasings and structures, reflecting the variety expected in practical applications. Note that there are seldom SQL queries with multiple corresponding NL queries in the BIRD dataset. Thus, we build the QVT dataset using Spider Dev set, as it contains 469 SQLs corresponding to more than two different NL queries, aligning with QVT's purpose. We compute the QVT scores based on the Equation (1).

As shown in Figure 8, there is no clear winner between LLM-based methods and PLM-based methods in terms of QVT. However, Fine-tuned LLMs generally exhibit higher QVT than prompting LLMs. This improvement may result from the alignment of model input with specific data distributions after fine-tuning, reducing the impact of NL changes on performance. Notably, although the Graphix+PICARD method underperforms in overall EX compared to all prompt-based methods, it surpasses them in QVT.

> **Finding 6.** *There is no clear winner between LLM-based and PLM-based methods in QVT. Fine-tuning the model with task-specific datasets may help stabilize its performance against NL variations.*
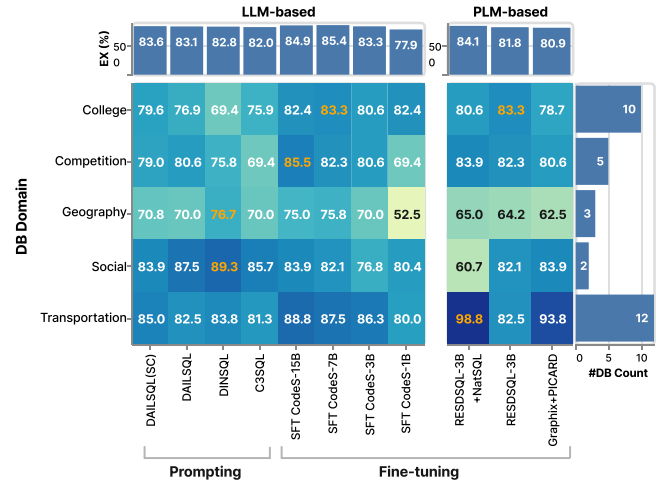


(a) Detailed Results
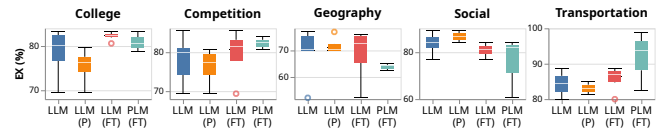


(b) Overall Results

Figure 9: EX *vs.* Different Domains on Spider.

**Exp-4: Database Domain Adaption.** In practical NL2SQL applications, scenarios usually involve domain-specific databases, such as movies or sports, each with unique schema designs and terminologies. Assessing the detailed performance of methods across different domains is crucial for effective model application. We classified the 140 databases in the Spider training set and the 20 databases in the development set into 33 domains. All fine-tuning-based LLMs and PLMs are tuned using the training set. Figure 9(a) shows the EX performance across diverse database domains in the Spider dataset. Figure 9(b) shows the overall performance.

As shown in Figure 9(a), we discovered that different NL2SQL methods exhibit varying biases towards different domains and there is no clear winner between LLM-based and PLM-based methods.

However, in Figure 9(b), we observe that fine-tuning-based methods outperform in domains with more training databases (College, Competition, Transportation). Conversely, in domains with fewer training databases, prompt-based methods excel. This suggests that in-domain training data during the fine-tuning process is crucial for enhancing model performance in specific domains.

> **Finding 7.** *Different methods exhibit varying biases towards different domains, and there is no clear winner between LLM-based and PLM-based methods. However, in-domain training data during fine-tuning is crucial for model performance in specific domains.*

**Exp-5: Supervised Fine-tuning of LLM-based Methods.** We investigated Supervised Fine-tuning (SFT) of *open-source* LLMs for the NL2SQL task. DAILSQL [11] examines the impact of varying shot and prompt representation during SFT but does not address *which open-source LLMs are best suited for SFT in the NL2SQL task.* DAILSQL found that SQL-style prompts were beneficial, so we adopted a similar prompt approach in a zero-shot setting, as shown

```
/* Given the following database schema: */
CREATE TABLE airports (
    City text,
    AirportCode text primary key,
    AirportName text
)

CREATE TABLE airlines (
    uid int primary key,
    Airline text
)

/* Answer the following: What are airport names at City
'Aberdeen'?*/
SELECT AirportName FROM airports WHERE City = "Aberdeen";
```

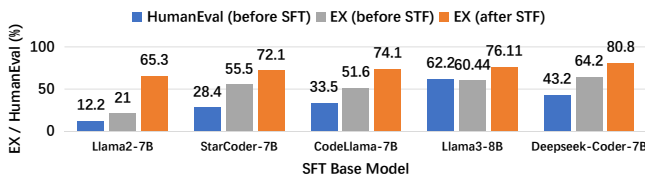**Figure 10: An Example of SQL-style Prompt.**



**Figure 11: EX / HumanEval vs. SFT Base Models.**

in Figure 10. Given that NL2SQL is a code-related task, we selected five open-source LLMs with varying code abilities, evaluated using the HumanEval (Pass@1) metric [5]. To ensure a fair comparison and account for hardware limitations, all chosen LLMs have similar parameters. The suffix in the model name, such as **7B**, indicates the model has 7 billion parameters.

*Settings.* We compare 5 fine-tuning-based LLMs introduced in Section 4.1. We use an instruction-tuning approach, *i.e.,* Alpaca [49]. We use the Adam optimizer with a learning rate of 1e-5 and no weight decay. The learning rate follows a cosine decay to zero by the end of training. We train with a global batch size of 16 for a single epoch to mitigate over-fitting risks. After SFT, LLMs are evaluated on the Spider Dev set using the EX metric.

*Results.* As shown in Figure 11, after SFT, the performance (EX) improves but varies significantly across different base models. Importantly, a positive correlation is observed between these performance variations and the models' intrinsic coding abilities (HumanEval) before SFT. This suggests that *selecting base LLMs with advanced coding capabilities is beneficial for adaptation in the NL2SQL task.*

Finding 8. *After Supervised Fine-tuning (SFT) on open-source LLMs for the NL2SQL task, we found a positive correlation between performance after SFT and the model's inherent coding ability prior to SFT. This indicates that base LLMs with advanced coding abilities are important for adapting to the NL2SQL task.*

## 4.3 Experiments on Evaluating Efficiency

**Exp-6: Economy of LLM-based Methods.** Prompt-based LLM methods utilize commercial GPT API interfaces to accomplish the NL2SQL task. As of June 2024, compared to GPT-3.5-turbo, the API interface of GPT-4 is 60 times more expensive for input tokens and 40 times more expensive for output tokens. In practical applications,

**Table 5: Accuracy vs. LLM Economy on Spider/BIRD Dev Set.**

| Methods | LLMs | Avg. Tokens / Query | | Avg. Cost / Query | | EX(%) | | EX / Avg. Cost | |
|---|---|---|---|---|---|---|---|---|---|
| | | Spider | BIRD | Spider | BIRD | Spider | BIRD | Spider | BIRD |
| C3SQL | GPT-3.5 | 5702 | 5890 | 0.0103 | 0.0104 | 82.0 | 50.2 | 7961 | 4825 |
| DINSQL | GPT-4 | 9571 | - | 0.2988 | - | 82.8 | - | 277 | - |
| DAILSQL | GPT-4 | 930 | 1559 | 0.0288 | 0.0486 | 83.1 | 54.3 | 2885 | 1117 |
| DAILSQL(SC) | GPT-4 | 1063 | 1886 | 0.0377 | 0.0683 | 83.6 | 55.9 | 2218 | 819 |
| **SuperSQL** | **GPT-4** | **942** | **1412** | **0.0354** | **0.0555** | **87.0** | **58.5** | **2458** | **1053** |

**Table 6: The Efficiency of PLM-based Methods.**

| Methods | Parameters | EX (%) | Latency Per Sample (sec) | GPU Memory Used (GiB) |
|---|---|---|---|---|
| RESDSQL-Base | 220M | 77.9 | 1.10 | 3.87 |
| RESDSQL-Base + NatSQL | 220M | 80.2 | 1.01 | 3.59 |
| RESDSQL-Large | 770M | 80.1 | 1.71 | 7.55 |
| RESDSQL-Large + NatSQL | 770M | 81.9 | 1.57 | 6.83 |
| RESDSQL-3B | 3B | 81.8 | 1.91 | 24.66 |
| RESDSQL-3B + NatSQL | 3B | 84.1 | 1.97 | 21.59 |

our concern extends beyond the performance of NL2SQL methods to include cost considerations. In this experiment, we compute several metrics for each prompt-based method based on the development set of Spider and BIRD. These include the number of tokens and the cost (in dollars) per NL2SQL task. As shown in Table 5, we also calculate the ratio of EX to Average Cost, which indicates the cost-effectiveness of the NL2SQL method to some extent.

Although C3SQL scores lowest in EX on both datasets, its EX to average cost ratio is the highest, benefiting from the lower cost of the GPT-3.5-turbo interface compared with GPT-4. Among methods using GPT-4, DINSQL is the least cost-effective, whereas DAILSQL emerges as the most cost-efficient. Although DAILSQL(SC) outperforms DAILSQL on both datasets, it introduces higher costs.

Finding 9. *Based on the ratio of Execution Accuracy (EX) to the Average Cost, we observe that prompt-based LLM methods calling GPT-3.5-turbo offer higher cost-effectiveness. Although DAILSQL(SC) shows EX improvements over DAILSQL on Spider and BIRD datasets, it introduces higher costs reducing its cost-effectiveness.*

**Exp-7: Efficiency of PLM-based Methods.** In practical applications, considering both the performance and efficiency of NL2SQL methods is essential, especially latency per sample. Different methods have varying GPU memory requirements, increasing with model size. Selecting the appropriate method based on available hardware and latency requirements is a common challenge. We assess three metrics across six models: RESDSQL-Base/Large/3B and RESDSQL-Base/Large/3B + NatSQL, focusing on Execution Accuracy (EX), Latency Per Sample, and GPU Memory Used, using the Spider development set. Model efficiency is dataset-agnostic, so we omit the BIRD dataset due to space constraints.

Table 6 shows that as model size increases, so do GPU memory and latency. However, RESDSQL-Base+NatSQL (220M parameters) and RESDSQL-Large (770M parameters) achieve similar EX (80.2% and 80.1%), with the former having lower latency and memory usage. Similarly, RESDSQL-Large+NatSQL and RESDSQL-3B have comparable EX but differ in latency and hardware needs. Therefore, model selection should consider latency and hardware resources.

Finding 10. *For the same method, as model parameters increase in size, there is a corresponding rise in the latency and hardware resource requirements. Furthermore, models with similar performance can differ in latency and hardware resource requirements.*

**Table 7: The Valid Efficiency Score Results.**

**(a) The Valid Efficiency Score in Spider-Dev.**

| Types | | Methods | Spider-Dev | | | | |
|---|---|---|---|---|---|---|---|
| | | | Easy | Medium | Hard | Extra | All |
| LLM-based | Prompting | C3SQL | 104.68 | 96.04 | 84.55 | 69.63 | 91.94 |
| | | DINSQL | 102.99 | 97.49 | 84.05 | 67.81 | 91.78 |
| | | DAILSQL | 102.73 | 100.36 | 86.15 | 66.10 | 93.04 |
| | | DAILSQL(SC) | 103.86 | 102.73 | 86.40 | 71.59 | 95.25 |
| | Fine-tuning | SFT CodeS-1B | 103.23 | 94.13 | 80.37 | 55.02 | 87.72 |
| | | SFT CodeS-3B | 106.17 | 99.72 | 80.80 | 68.10 | 93.01 |
| | | SFT CodeS-7B | 108.77 | 102.90 | 84.05 | 73.42 | 96.41 |
| | | SFT CodeS-15B | 107.91 | 103.02 | 87.10 | 68.92 | 96.04 |
| PLM-based | | RESDSQL-3B | 106.22 | 98.61 | 83.06 | 61.60 | 91.88 |
| | | RESDSQL-3B + NatSQL | 106.91 | 97.98 | 86.78 | 73.83 | 94.36 |
| | | Graphix + PICARD | 108.92 | 102.71 | 83.64 | 68.61 | 95.51 |
| Hybird | | SuperSQL | 107.54 | 104.32 (1.30↑) | 96.98 (9.88↑) | 75.18 (1.35↑) | 99.18 (2.77↑) |

**(b) The Valid Efficiency Score in BIRD-Dev.**

| Types | | Methods | BIRD-Dev | | | |
|---|---|---|---|---|---|---|
| | | | Simple | Moderate | Challenging | All |
| LLM-based | Prompting | C3SQL | 59.82 | 41.68 | 31.93 | 51.70 |
| | | DAILSQL | 65.04 | 43.35 | 39.33 | 56.05 |
| | | DAILSQL(SC) | 66.54 | 46.14 | 45.18 | 58.35 |
| | Fine-tuning | SFT CodeS-1B | 61.11 | 39.89 | 37.38 | 52.45 |
| | | SFT CodeS-3B | 64.96 | 50.98 | 38.99 | 58.28 |
| | | SFT CodeS-7B | 66.88 | 49.53 | 58.42 | 60.83 |
| | | SFT CodeS-15B | 67.87 | 51.69 | 52.71 | 61.54 |
| PLM-based | | RESDSQL-Base | 42.75 | 22.16 | 16.54 | 34.05 |
| | | RESDSQL-Large | 47.21 | 30.00 | 34.67 | 40.81 |
| | | RESDSQL-3B | 53.35 | 35.49 | 28.84 | 45.64 |
| Hybird | | SuperSQL | 69.75 (1.88↑) | 50.55 | 49.08 | 61.99 (0.45↑) |

**Exp-8: SQL Efficiency - Valid Efficiency Score.** In practical scenarios, it's crucial not only to focus on the correctness of the SQL queries generated by models but also on their execution efficiency. BIRD [26] introduces the Valid Efficiency Score (VES) to evaluate the execution efficiency of correctly generated SQL queries. The VES score is determined by dividing the execution time of the ground truth SQL query by the execution time of the predicted SQL query. We evaluate different methods on the development set of Spider and BIRD using the VES metric to compare the execution efficiency of SQL generated by different methods.

Table 7 reports experimental results. The highest VES score is highlighted in orange in the table. The methods with the highest VES on subsets of varying difficulty are inconsistent, with no clear advantage for LLM-based or PLM-based approaches. Generally, a method shows lower VES on harder subsets, likely due to increased complexity and longer execution time.

> Finding 11. Based on the VES metric, there is no clear winner between LLM-based and PLM-based methods. For the same method, it tends to have lower VES on more difficult subsets.
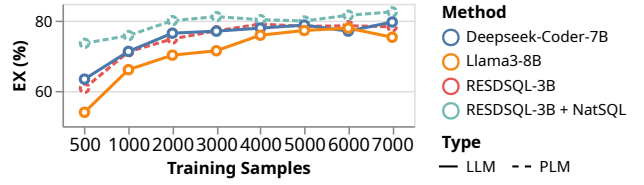


**Figure 12: EX vs. #-Training Samples on Spider.**

**Exp-9: The Impact of the #-Training Samples.** In real-world scenarios, limited in-domain data often hinders performance. We experiment on the Spider training set, sampling subsets with size increments of 1000 and a smaller subset of 500. We train various methods on these subsets and evaluate their EX performance on the Spider development set. The training hyper-parameters for RESDSQL-3B and RESDSQL-3B+NatSQL match those in [23], while other methods follow **Exp-5**.

The results in Figure 12 show that both PLM-based and fine-tuned LLM methods improve with more NL2SQL training data and achieve acceptable performance with 4000 training samples. However, the EX performance gains decrease as dataset size increases.

> Finding 12. *Both PLM-based and LLM-based methods improve with more NL2SQL training data. However, the EX performance gains decrease as dataset size increases. If data privacy is a concern or sufficient labeled data is available, fine-tuning LLM/PLM is promising.*

## 5 COMBINING THE BEST OF BOTH WORLDS
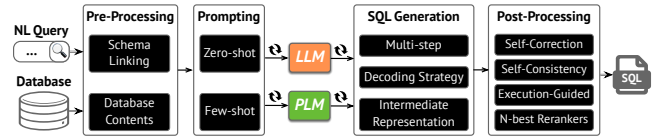
### 5.1 A Design Space Exploration



**Figure 13: The Design Space of the NL2SQL Solution.**

We explore the design space of NL2SQL solution powered by language models, as shown in Figure 13.

(1) **Pre-Processing**: The Pre-Processing module comprises schema linking and DB contents. Schema linking maps NL query references to database schema elements (tables, columns), enhancing cross-domain generalizability and complex query generation [20]. This approach is adopted by leading LLM-based [9, 38] and PLM-based methods [23, 25]. Additionally, the DB content module aligns query conditions with database content, often enriching column details via string matching [28]. As detailed in Table 1, while prevalent in PLM-based methods, it's seldom utilized in LLM-based approaches.

(2) **Prompting Strategy**: Prompting strategies fall into zero-shot, where no NL2SQL examples are included in the model input, and few-shot, which incorporates such examples, denoted as "3-shot", "5-shot", etc., depending on the number of examples used. Table 1 shows PLM-based methods typically use zero-shot, while LLM-based methods vary: C3SQL [9] employs zero-shot, whereas DAILSQL[11] and DINSQL[38] use few-shot. The few-shot examples for DINSQL are manually designed and fixed, whereas those for

DAILSQL are dynamically selected based on the similarity between the target question and training set examples.

(3) **SQL Generation Strategy**: Language models employ various strategies for generating SQL, categorized into three key aspects: Multi-Step, Decoding Strategy, and Intermediate Representation.

(a) *Multi-Step* akin to the Chain-of-Thought (COT) process, involves generating SQL queries in stages, particularly useful for complex queries [54]. We include two types of multi-step strategies: "SQL skeleton - SQL" from PLM-based RESDSQL [23] and "Subquery - SQL" from DINSQL [38].

(b) *Decoding Strategy* involves the model's decoding process to ensure output validity. The PLM-based PICARD [45] enforces SQL syntax compliance in its output, whereas LLM-based methods, utilizing OpenAI's API, lack this decoding-level restriction.

(c) *Intermediate Representation* strategy explores if a model employs an intermediary query form to address the NL to SQL translation's *mismatch problem*, where SQL's design for relational databases doesn't align with natural language semantics. Various solutions like [15] and NatSQL [10] have been introduced. LLM-based DINSQL [38] and several PLM-based methods [10, 23, 42] adopt NatSQL. In our setting, we only include NatSQL for simplification.

(4) **Post-Processing**: we consider the following strategies.

(a) *Self-Correction* is proposed in DINSQL [38]. It provides the generated SQL to the model for fixing potential issues.

(b) *Self-Consistency* involves executing various valid SQL queries for a single NL query, using a voting mechanism on the outcomes to determine the most consistent SQL as the final choice. It is used in C3SQL [9] and DAILSQL [11].

(c) *Execution-Guided SQL Selector* is a module [23] that sequentially executes model-generated SQL queries, identifying the first error-free execution as the valid SQL.

(d) *N-best Rerankers* rank multiple candidate SQL queries to select the most probable one as the final query [59].

## 5.2 NL2SQL360 Facilitates Better NL2SQL

After categorizing different LLM- and PLM-based methods into a unified modular framework, a question arises: *Can automatically exploring and combining different modules from various NL2SQL solutions achieve stronger performance?*

To address this question, inspired by the Neural Architecture Search (NAS) algorithm [55], we designed an NL2SQL Automated Architecture Search algorithm (**NL2SQL360-AAS**) within our NL2SQL360 framework. The key intuition behind **NL2SQL360-AAS** is to automatically explore the predefined design space (*i.e.,* predefined search space) of the NL2SQL solution. Therefore, we adopt the standard Genetic Algorithm (GA) [3] to achieve this goal.

There are some key concepts relevant to our **NL2SQL360-AAS**.

(1) *Search Space*. This includes various modules used in NL2SQL, such as SQL generation strategies, post-processing modules, and prompting techniques, as shown in Figure 13.

(2) *Individual*. A valid combination of different modules in the search space, *i.e.,* a valid NL2SQL solution, is an *individual*.

(3) *Target Metrics*. We aim to select better individuals based on target metrics like Execution Accuracy (EX), Exact-Match Accuracy (EM), and Valid Efficiency Score (VES) on a specified dataset.
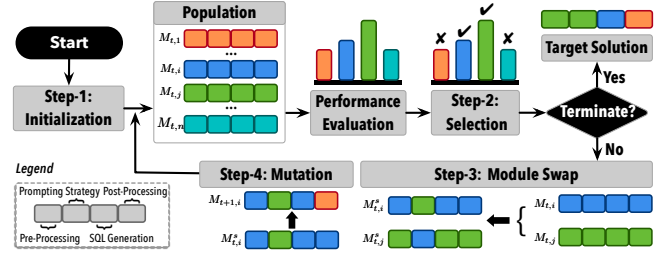


**Figure 14: NL2SQL360-AAS Algorithm Overview.**

**NL2SQL360-AAS: An Overview.** As shown in Figure 14, our algorithm consists of four main steps, *i.e.,* Initialization, Individual Selection, NL2SQL Module Swap, and NL2SQL Module Mutation. Note that, $M_{t,i}$ is the $i$-th individual in the $t$-th generation population.

*Step-1: Initialization.* We initialize $N$ randomized NL2SQL system individuals $\{M_{0,n}\}_{n=1}^{N}$ that are composed of random modules as shown in Figure 13, resulting in 0-th generation population.

*Step-2: Individual Selection.* We evaluate the population of $N$ individuals on the specified dataset (*e.g.,* Spider) using the target metric (*e.g.,* EX). We implement a *Russian Roulette Process* [55] for individual selection. This process probabilistically samples individuals based on their target metric distribution, ensuring that individuals with higher target metrics have a greater likelihood of being selected, while consistently eliminating the lowest performers.

*Step-3: NL2SQL Module Swap.* Two individuals selected from the previous step will exchange their NL2SQL modules based on the module swap probability $p_s$. For example, if individual $M_{t,i}$ has a Self-Correction module and individual $M_{t,j}$ has a Self-Consistency module in the Post-Processing Layer before the swap, these two modules could be exchanged. In Figure 14, the individuals after the module swap are labeled as $M_{t,i}^s$ and $M_{t,j}^s$, respectively.

*Step-4: NL2SQL Module Mutation.* Next, the individual $M_{t,i}^s$ (similarly $M_{t,j}^s$) will undergo module mutation in each layer (*e.g.,* Pre-Processing Layer) based on the module mutation probability $p_m$. For example, if the Pre-Processing Layer of $M_{t,i}^s$ does not use the DB Contents module, a successful mutation will result in the inclusion of this module. After mutation, the individual is labeled as $M_{t+1,i}$ and will enter the next generation population.

We repeat *Step* 2–4 until we obtain the complete next generation population $\{M_{t+1,n}\}_{n=1}^{N}$, marking one entire population iteration.

## 5.3 SuperSQL: A Superior NL2SQL Solution Suggested by NL2SQL360-AAS

In this section, we validate the effectiveness of the **NL2SQL360-AAS** algorithm. The search space is defined as shown in Figure 13. For simplification, we utilize only the Few-shot module from DAILSQL in the Prompting Strategy. Since we use GPT as our backbone, we cannot adjust its decoding strategy and thus we can only use Greedy Search in the Decoding phase. To save costs, we use GPT-3.5-turbo in the algorithm. The population size $N$ is set to 10, the number of population generations $T$ is 20, and the probabilities for NL2SQL module swap and NL2SQL module mutation, $p_s$ and $p_m$, are

```
/* Given the following database schema (with linked
value list after column definition): */
CREATE TABLE airports (
    City text,
    AirportCode text primary key,
    AirportName text -- ["Aberdeen"]
)

CREATE TABLE airlines (
    uid int primary key,
    Airline text
}

/* Answer the following: What are airport names at City
'Aberdeen'?*/
SELECT AirportName FROM airports WHERE City = "Aberdeen";
```

**Figure 15: Clear Schema with DB Content Prompt.**

set to 0.5 and 0.2, respectively. We run the algorithm on the Spider development set with the EX as the target metric.

**SuperSQL.** We select the individual with the highest Execution Accuracy from the final generation produced by the NL2SQL360-AAS as our final searched NL2SQL solution, namely **SuperSQL**.

The composition of **SuperSQL** is as follows: (1) Pre-Processing: Schema Linking from RESDSQL and DB Contents from BRIDGE v2; (2) Prompting: DAILSQL's Few-shot module selects in-context examples by similarity; (3) SQL Generation: OpenAI's Greedy-decoding strategy, without Multi-step or NatSQL; (4) Post-Processing: Self-Consistency from DAILSQL. We explored the organization of the prompt for this composition, as illustrated in Figure 15. Under this combination, the RESDSQL schema linking module is used to filter out irrelevant schema items. Furthermore, it incorporates the DB content module from the BRIDGE v2 method. The relevant content is added as comments next to the columns in the prompt, enhancing column information. We then replace the backbone model with GPT-4 for improved performance.

**The Effectiveness of SuperSQL.** We evaluate **SuperSQL** on the Spider development set, achieving 87.0% in EX and outperforming other methods (Table 3). **SuperSQL** excels in Medium, Hard, and Extra subsets, proving its effectiveness. On the BIRD development set, **SuperSQL** also shows competitive performance (Table 4).

We also evaluate **SuperSQL** on the Spider and BIRD test sets, achieving 87.0% EX on Spider (2nd place) and 62.66% EX on BIRD (9th place). Note that **SuperSQL** surpasses all baselines within its design space. Specifically, **SuperSQL** outperforms the strongest *baseline*—DAILSQL(SC)—by 5.25% on the BIRD test set. This improvement is primarily due to our **NL2SQL360-AAS**, which effectively searches for superior module combinations based on different *baselines* in the design space. Adding more powerful *baselines* is expected to further improve **SuperSQL** through **NL2SQL360-AAS**.

**The Efficiency of SuperSQL.** We calculate the VES metric to evaluate SQL efficiency on the development set of Spider and BIRD. According to Table 7, **SuperSQL** attains overall VES scores of 99.18 and 61.99, respectively, outperforming other methods.

**The Economy of SuperSQL.** Furthermore, we consider the economy of our method, and the results are shown in Table 5. Compared to other GPT-4 based methods, our method uses fewer tokens and lower costs, while achieving better performance in EX metrics.

## 6 RESEARCH OPPORTUNITIES

We discuss research opportunities based on experimental findings.

**Make NL2SQL Methods Trustworthy.** Current methods may generate incorrect SQL results, which can be attributed to: 1) ambiguous and underspecified NL queries, 2) ambiguous database schemas and dirty contents, and 3) inadequate capabilities in schema linking.

*Handling ambiguous and underspecified NL queries.* We can explore the following strategies to alleviate these issues. *(i) Query Rewriter* aims to automatically refine given NL queries and ensure their clarity. *(ii) Query Auto-completion* helps formulate user queries by suggesting candidate tokens that are well-aligned with the database.

*Interpreting NL2SQL Solution. (i) NL2SQL Debugger* detects incorrect SQL queries, enables users to step through the SQL generation process, and helps identify errors or mismatches. *(ii) SQL and Query Results Interpretation* method helps users assess if the generated SQL and query results meet their requirements.

**Develop Cost-effective NL2SQL Methods.** LLM-based NL2SQL methods are promising but costly in terms of token consumption, impacting both costs and inference times. Exploring ways to enhance accuracy while minimizing token use is crucial. Specifically, the potential benefits of modularized NL2SQL solutions and multi-agent frameworks are becoming clear. Incorporating LLMs with these methods has the potential to optimize both accuracy and efficiency, particularly for complex queries, while conserving tokens.

**Adaptive Training Data Generation.** The effectiveness of NL2SQL methods depends greatly on the quality and coverage of training data. These methods often struggle with adapting to unseen databases. A promising research direction is dynamically generating (NL, SQL) pairs using model evaluation feedback. This approach tackles domain adaptation and ensures diverse, high-quality training data by leveraging insights from NL2SQL performance.

## 7 CONCLUSION

We proposed a multi-angle testbed, named **NL2SQL360**, for evaluating NL2SQL methods from different perspectives, such as the ability to handle various characteristics of SQL and database domains, in a fine-grained manner. We utilized our **NL2SQL360** to evaluate 13 LLM-based and 7 PLM-based NL2SQL methods on 2 widely-used benchmarks, varying 15 settings and deriving a set of new findings. Furthermore, we employed our **NL2SQL360** to analyze the design space for NL2SQL solutions and automatically search for one of the best solutions, named **SuperSQL**, tailored to user-specific needs. Our new **SuperSQL**, which interleaves LLM-based and PLM-based modules, achieves **87%** and **62.66%** execution accuracy on the Spider and BIRD test sets, respectively.

## ACKNOWLEDGMENTS

# REFERENCES

[1] 2024. TABLEAU SOFTWARE, LLC, A SALESFORCE COMPANY. https://www.tableau.com/. Accessed: 2024-2-22.

[2] AI@Meta. 2024. Llama 3 Model Card. (2024). https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[3] Tanweer Alam, Shamimul Qamar, Amit Dixit, and Mohamed Benaida. 2020. Genetic Algorithm: Reviews, Implementations, and Applications. *CoRR* abs/2007.12673 (2020). arXiv:2007.12673 https://arxiv.org/abs/2007.12673

[4] 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*. OpenReview.net.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[6] Ziru Chen, Shijie Chen, Michael White, Raymond J. Mooney, Ali Payani, Jayanth Srinivasa, Yu Su, and Huan Sun. 2023. Text-to-SQL Error Correction with Language Models of Code. In *ACL (2)*. 1359–1372.

[7] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 1337–1350. https://doi.org/10.18653/V1/2021.NAACL-MAIN.105

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[9] 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. *CoRR* abs/2307.07306 (2023). https://doi.org/10.48550/ARXIV.2307.07306 arXiv:2307.07306

[10] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John H. 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, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 2030–2042. https://doi.org/10.18653/V1/2021.FINDINGS-EMNLP.174

[11] 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 (2024), 1132–1145. https://www.vldb.org/pvldb/vol17/p1132-gao.pdf

[12] Orest Gkini, Theofilos Belmpas, Georgia Koutrika, and Yannis E. Ioannidis. 2021. An In-Depth Benchmarking of Text-to-SQL Systems. In *SIGMOD Conference*. ACM, 632–644.

[13] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. *Proc. ACM Manag. Data* 1, 2 (2023), 147:1–147:28.

[14] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming - The Rise of Code Intelligence. *CoRR* abs/2401.14196 (2024). https://doi.org/10.48550/ARXIV.2401.14196 arXiv:2401.14196

[15] 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. *CoRR* abs/1905.08205 (2019). arXiv:1905.08205 http://arxiv.org/abs/1905.08205

[16] Yiqun Hu, Yiyun Zhao, Jiarong Jiang, Wuwei Lan, Henghui Zhu, Anuj Chauhan, Alexander Hanbo Li, Lin Pan, Jun Wang, Chung-Wei Hang, Sheng Zhang, Jiang Guo, Mingwen Dong, Joseph Lilien, Patrick Ng, Zhiguo Wang, Vittorio Castelli,

and Bing Xiang. 2023. Importance of Synthesizing High-quality Data for Text-to-SQL Parsing. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 1327–1343. https://doi.org/10.18653/V1/2023.FINDINGS-ACL.86

[17] George Katsogiannis-Meimarakis and Georgia Koutrika. 2021. A Deep Dive into Deep Learning Approaches for Text-to-SQL Systems. In *SIGMOD Conference*. ACM, 2846–2851.

[18] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *VLDB J.* 32, 4 (2023), 905–936.

[19] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleD-BQA: Realistic Evaluation of Text-to-SQL Parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 2261–2273. https://doi.org/10.18653/V1/2021.ACL-LONG.176

[20] 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

[21] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. 7871–7880.

[22] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 709–712. https://doi.org/10.1145/2588555.2594519

[23] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 13067–13075. https://doi.org/10.1609/AAAI.V37I11.26535

[24] 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. *CoRR* abs/2402.16347 (2024). https://doi.org/10.48550/ARXIV.2402.16347 arXiv:2402.16347

[25] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-T5: Mixing Pre-trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 13076–13084. https://doi.org/10.1609/AAAI.V37I11.26536

[26] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan 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. *CoRR* abs/2305.03111 (2023). https://doi.org/10.48550/ARXIV.2305.03111 arXiv:2305.03111

[27] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *CoRR* abs/2305.06161 (2023). https://doi.org/10.48550/ARXIV.2305.06161 arXiv:2305.06161

[28] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. In *Findings*

*of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.), Vol. EMNLP 2020. Association for Computational Linguistics, 4870–4888. https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.438

[29] Aiwei Liu, Xuming Hu, Li Lin, and Lijie Wen. 2022. Semantic Enhanced Text-to-SQL Parsing via Iteratively Learning Schema Linking Graph. In *KDD*. ACM, 1021–1030.

[30] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Yuyu Luo, and Nan Tang. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? (2024). https://github.com/HKUSTDial/NL2SQL_Handbook

[31] Yuyu Luo, Xuedi Qin, Chengliang Chai, Nan Tang, Guoliang Li, and Wenbo Li. 2022. Steerable Self-Driving Data Visualization. *IEEE Trans. Knowl. Data Eng.* 34, 1 (2022), 475–490.

[32] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. DeepEye: Towards Automatic Data Visualization. In *ICDE*. IEEE Computer Society, 101–112.

[33] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. DeepEye: Creating Good Data Visualizations by Keyword Search. In *SIGMOD Conference*. ACM, 1733–1736.

[34] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *SIGMOD Conference*. ACM, 1235–1247.

[35] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2022. Natural Language to Visualization by Neural Machine Translation. *IEEE Trans. Vis. Comput. Graph.* 28, 1 (2022), 217–226.

[36] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large Language Models: A Survey. *CoRR* abs/2402.06196 (2024). https://doi.org/10.48550/ARXIV.2402.06196 arXiv:2402.06196

[37] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). https://doi.org/10.48550/ARXIV.2303.08774 arXiv:2303.08774

[38] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html

[39] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. RASAT: Integrating Relational Structures into Pretrained Seq2Seq Model for Text-to-SQL. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, 3215–3229. https://doi.org/10.18653/V1/2022.EMNLP-MAIN.211

[40] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[41] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[42] Daking Rai, Bailin Wang, Yilun Zhou, and Ziyu Yao. 2023. Improving Generalization in Language Model-based Text-to-SQL Semantic Parsing: Two Simple Semantic Boundary-based Techniques. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 150–160. https://doi.org/10.18653/V1/2023.ACL-SHORT.15

[43] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the Text-to-SQL Capabilities of Large Language Models. *CoRR* abs/2204.00498 (2022).

[44] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. *CoRR* abs/2308.12950 (2023). https://doi.org/10.48550/ARXIV.2308.12950 arXiv:2308.12950

[45] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 9895–9901.

[46] Leixian Shen, Enya Shen, Yuyu Luo, Xiaocong Yang, Xuming Hu, Xiongshuai Zhang, Zhiwei Tai, and Jianmin Wang. 2021. Towards Natural Language Interfaces for Data Visualization: A Survey. *CoRR* abs/2109.03506 (2021).

[47] Jiawei Tang, Yuyu Luo, Mourad Ouzzani, Guoliang Li, and Hongyang Chen. 2022. Sevi: Speech-to-Visualization through Neural Machine Translation. In *SIGMOD Conference*. ACM, 2353–2356.

[48] Nan Tang, Chenyu Yang, Ju Fan, Lei Cao, Yuyu Luo, and Alon Y. Halevy. 2024. VerifAI: Verified Generative AI. In *CIDR*. www.cidrdb.org.

[49] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[50] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023). https://doi.org/10.48550/ARXIV.2307.09288 arXiv:2307.09288

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[52] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. *CoRR* abs/2312.11242 (2023). https://doi.org/10.48550/ARXIV.2312.11242 arXiv:2312.11242

[53] Lihan Wang, Bowen Qin, Binyuan Hui, Bowen Li, Min Yang, Bailin Wang, Binhua Li, Jian Sun, Fei Huang, Luo Si, and Yongbin Li. 2022. Proton: Probing Schema Linking Information from Pre-trained Language Models for Text-to-SQL Parsing. In *KDD*. ACM, 1889–1898.

[54] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR* abs/2201.11903 (2022). arXiv:2201.11903 https://arxiv.org/abs/2201.11903

[55] Lingxi Xie and Alan L. Yuille. 2017. Genetic CNN. *CoRR* abs/1703.01513 (2017). arXiv:1703.01513 http://arxiv.org/abs/1703.01513

[56] Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. HAIChart: Human and AI Paired Visualization System. *Proc. VLDB Endow.* 17.

[57] Yilin Ye, Jianing Hao, Yihan Hou, Zhan Wang, Shishi Xiao, Yuyu Luo, and Wei Zeng. 2024. Generative AI for visualization: State of the art and future directions. *Visual Informatics* 8, 2 (2024), 43–66. https://doi.org/10.1016/j.visinf.2024.04.003

[58] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*. Association for Computational Linguistics, 3911–3921.

[59] Lu Zeng, Sree Hari Krishnan Parthasarathi, and Dilek Hakkani-Tur. 2022. N-Best Hypotheses Reranking for Text-to-SQL Systems. In *IEEE Spoken Language Technology Workshop, SLT 2022, Doha, Qatar, January 9-12, 2023*. IEEE, 663–670. https://doi.org/10.1109/SLT54892.2023.10023434

[60] 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. 2023. A Survey of Large Language Models. *CoRR* abs/2303.18223 (2023). https://doi.org/10.48550/ARXIV.2303.18223 arXiv:2303.18223

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

[62] Yizhang Zhu, Shiyin Du, Boyan Li, Yuyu Luo, and Nan Tang. 2024. Are Large Language Models Good Statisticians? *CoRR* abs/2406.07815 (2024).