OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale

Haoyang Li
Renmin University of China
lihaoyang.cs@ruc.edu.cn

Shang Wu
Renmin University of China
wushang_@ruc.edu.cn

Xiaokang Zhang
Renmin University of China
zhang2718@ruc.edu.cn

Xinmei Huang
Renmin University of China
huangxinmei@ruc.edu.cn

Jing Zhang
Renmin University of China
zhang-jing@ruc.edu.cn

Fuxin Jiang
ByteDance Inc
jiangfuxin@bytedance.com

Shuai Wang
ByteDance Inc
wangshuai.will@bytedance.com

Tieying Zhang
ByteDance Inc
tieying.zhang@bytedance.com

Jianjun Chen
ByteDance Inc
jianjun.chen@bytedance.com

Rui Shi
ByteDance Inc
shirui@bytedance.com

Hong Chen
Renmin University of China
chong@ruc.edu.cn

Cuiping Li
Renmin University of China
licuiping@ruc.edu.cn

ABSTRACT

Text-to-SQL, the task of translating natural language questions into SQL queries, plays a crucial role in enabling non-experts to interact with databases. While recent advancements in large language models (LLMs) have significantly enhanced text-to-SQL performance, existing approaches face notable limitations in real-world text-to-SQL applications. Prompting-based methods often depend on closed-source LLMs, which are expensive, raise privacy concerns, and lack customization. Fine-tuning-based methods, on the other hand, suffer from poor generalizability due to the limited coverage of publicly available training data. To overcome these challenges, we propose a novel and scalable text-to-SQL data synthesis framework for automatically synthesizing large-scale, high-quality, and diverse datasets without extensive human intervention. Using this framework, we introduce SynSQL-2.5M, the first million-scale text-to-SQL dataset, containing 2.5 million samples spanning over 16,000 synthetic databases. Each sample includes a database, SQL query, natural language question, and chain-of-thought (CoT) solution. Leveraging SynSQL-2.5M, we develop OmniSQL, a powerful open-source text-to-SQL model available in three sizes: 7B, 14B, and 32B. Extensive evaluations across nine datasets demonstrate that OmniSQL achieves state-of-the-art performance, matching or surpassing leading closed-source and open-source LLMs, including GPT-4o and DeepSeek-V3, despite its smaller size. We release all code, datasets, and models to support further research.

PVLDB Reference Format:
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. OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale. PVLDB, 18(11): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

Jing Zhang and Tieying Zhang are the corresponding authors.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/RUCKBReasoning/OmniSQL.

1 INTRODUCTION

Text-to-SQL translates natural language (NL) questions into executable SQL queries, enabling non-experts to interact with databases effectively [1, 40, 47]. This capability supports a wide range of data-centric applications and has garnered significant research interest from both natural language processing (NLP) and database communities [18, 21, 24, 42–44, 59].

State-of-the-Art: Strengths and Limitations. Recent advancements in large language models (LLMs) have driven significant progress in the text-to-SQL field. State-of-the-art (SOTA) solutions [22, 58] often employ multi-agent collaborative frameworks, where specialized agents tackle distinct sub-tasks such as schema linking, text-to-SQL generation, SQL refinement, and SQL selection. Among these, text-to-SQL generation remains the core component. Current approaches to this component mainly rely on LLMs and can be broadly categorized into two paradigms: prompting-based and fine-tuning-based.

Prompting-based methods leverage powerful LLMs through carefully crafted prompts, often relying on closed-source models accessed via APIs. In contrast, fine-tuning-based methods focus on training LLMs on existing text-to-SQL datasets, such as Spider [87] and BIRD [45], to adapt them for the task. While both approaches have demonstrated impressive benchmark performance, they face notable limitations in real-world applications. Prompting-based methods suffer from challenges such as high usage costs, data privacy concerns, and limited control over model behavior due to their reliance on calling APIs. On the other hand, fine-tuning-based methods often struggle with generalizability to complex problems or domain-specific databases, as publicly available datasets provide

limited coverage of real-world scenarios. For instance, experiments reveal that Qwen2.5-Coder-7B-Instruct [30], fine-tuned on the training sets of Spider and BIRD, performs well on their development sets (which share the similar distribution as the training data) but achieves only 43.8% and 31.4% execution accuracy on out-of-domain datasets, ScienceBenchmark [88] and EHRSQL [38], respectively. In comparison, zero-shot prompting GPT-4-Turbo [54] achieves significantly higher execution accuracy of 59.2% and 43.1%, highlighting the limited generalizability of current fine-tuning approaches.

To address these limitations, enhancing the text-to-SQL capabilities of open-source LLMs through large-scale, diverse, high-quality training data presents a promising direction. Such efforts could improve both the performance and generalizability of open-source models. Moreover, open-source models can be deployed locally, making text-to-SQL systems more cost-effective, data-secure, and adaptable to modifications, thereby overcoming the challenges associated with closed-source LLMs. However, acquiring large-scale data through human annotation is often infeasible. To mitigate this, several early data augmentation methods [28, 76, 80, 83] have been proposed to expand existing text-to-SQL datasets. *Unfortunately, most of these methods focus on generating data samples that conform to the distribution of existing datasets, resulting in limited diversity, quality, and scalability. Additionally, many approaches require extensive human effort to define complex templates or grammars, further constraining their practicality.*

Our Proposal. To overcome these limitations, we propose a novel text-to-SQL data synthesis framework that distinguishes itself from existing data augmentation methods by offering the following key advantages: **(1) Automatic:** The entire synthesis process requires minimal to no human intervention. **(2) Scalable:** The framework can generate a large scale of diverse, high-quality data samples, ensuring coverage across a wide range of domains. **(3) Realistic:** The synthesized data aligns with real-world user needs and scenarios.

Challenges and Solutions. Designing an automatic, scalable, and realistic data synthesis framework is a non-trivial task. The primary challenge lies in ensuring automation and scalability while maintaining the quality and diversity of the generated data. To address this, we introduce a progressive pipeline that decouples the synthesis process into several simpler, manageable steps. Each step is automated using LLMs, minimizing human intervention. (1) The pipeline begins by leveraging web tables to synthesize realistic databases. Specifically, given a web table, the LLM is prompted to infer a plausible database business scenario associated with the table and generate a corresponding database. This synthetic database includes multiple relational tables with primary and foreign key relationships. Each relational table contains metadata such as the table name, description, column names, data types, column descriptions, and two example data rows. With millions of web tables available online [17], this approach ensures scalability across a wide range of domains. (2) Next, we generate meaningful SQL queries based on the synthesized databases by providing the LLM with the database information. (3) Then, we employ a back-translation technique to convert these SQL queries into semantically equivalent natural language questions. This technique, widely adopted in prior studies [4, 25, 28, 76, 80, 88, 91], could guarantee the quality of the synthetic <question, SQL query> pairs because converting SQL queries into natural language is inherently more accurate and less ambiguous than the reverse. (4) Finally, to bridge the gap between questions and SQL queries, we draw inspiration from chain-of-thought (CoT) reasoning [35, 78]. For each synthetic text-to-SQL data, we generate a step-by-step CoT solution that details the intermediate reasoning steps required to construct the SQL query from the question and the database. This not only enhances the interpretability of the synthetic data but also provides high-quality training signals for text-to-SQL models.

The second challenge is ensuring that the synthetic data aligns with real-world user needs and scenarios. A robust text-to-SQL model must accommodate a wide range of SQL queries, from simple to highly complex, reflecting both basic data retrieval and advanced data analysis requirements. To meet this demand, we define four levels of SQL complexity: simple, moderate, complex, and highly complex. During the SQL synthesis process, we select a complexity level and instruct the LLM to generate SQL queries that correspond to that level. Moreover, real-world users often express their questions in diverse linguistic styles, ranging from formal and explicit to vague and metaphorical. To address this variability, we identify nine common natural language styles: formal, colloquial, imperative, interrogative, descriptive, concise, vague, metaphorical, and conversational. When translating SQL queries into natural language questions, we adopt a specific style and guide the LLM to generate questions consistent with that style. This approach ensures that the synthetic data can accurately capture the various ways users might express their questions in real-world scenarios.

Validation. To validate the effectiveness of our proposed framework, we introduce SynSQL-2.5M, the first million-scale text-to-SQL dataset. Specifically, SynSQL-2.5M contains 2,544,390 text-to-SQL data samples, each represented as a quadruple of <database, question, SQL query, CoT solution>, spanning 16,583 distinct synthetic databases. Extensive statistics reveal that SynSQL-2.5M demonstrates high diversity and complexity compared to existing text-to-SQL datasets. We further evaluate its quality across four dimensions: database, question, SQL query, and data sample. When compared to the widely adopted human-labeled dataset BIRD [45], SynSQL-2.5M outperforms in nearly all criteria, underscoring the reliability and effectiveness of our data synthesis pipeline.

Building on SynSQL-2.5M, we develop OmniSQL, a powerful open-source text-to-SQL model available in three scales: 7B, 14B, and 32B. We evaluate OmniSQL across nine datasets, including three standard datasets (Spider development and test sets [87] and BIRD development set [45]), three domain-specific datasets (Spider2.0-SQLite [39], ScienceBenchmark [88], and EHRSQL [38]), and three robustness datasets (Spider-DK [20], Spider-Syn [19], and Spider-Realistic [12]). The results indicate that OmniSQL achieves state-of-the-art average performance across these datasets, matching or outperforming leading open-source LLMs (*e.g.*, DeepSeek-V3 [10] and Qwen2.5-72B-Instruct [81]) and advanced closed-source models (*e.g.*, GPT-4-Turbo [54] and GPT-4o [56]), despite its smaller model size. Our contributions are summarized as follows:

- **Data Synthesis Framework.** We propose an automatic and scalable framework for text-to-SQL data synthesis, addressing the generation of realistic databases, complexity-aware SQL queries, stylized natural language questions, and reliable chain-of-thought solutions.

- **Synthetic Dataset and Fine-tuned Multi-scale Model.** We introduce SynSQL-2.5M, the first million-scale text-to-SQL dataset including 2,544,390 diverse and high-quality data samples. Using SynSQL-2.5M, we fine-tune OmniSQL, a new text-to-SQL model available in three scales: 7B, 14B, and 32B.
- **New SOTA Text-to-SQL Performance.** Extensive experiments demonstrate that OmniSQL achieves new state-of-the-art performance in text-to-SQL tasks, surpassing leading open-source and closed-source LLMs with significantly fewer parameters, highlighting the effectiveness of our data synthesis framework. We have open-sourced our code, datasets, and models on GitHub[1] to facilitate further research in text-to-SQL.

2 RELATED WORK

2.1 Text-to-SQL

In the field of text-to-SQL, early studies typically employ an explicit encoder-decoder architecture. The encoder encodes the database schema and the question, while the decoder generates the corresponding SQL query based on the encoded information. Some studies enhance encoders by incorporating graph relation information [6, 7, 44, 75] or by leveraging pre-training techniques [12, 84, 85]. Other approaches focus on improving decoders by introducing grammar constraints, which help reduce syntax errors in generated SQL queries and thus improve the model's accuracy [69, 77].

With the rapid advancement of sequence-to-sequence (seq2seq) models, the text-to-SQL task has been transferred to a seq2seq modeling task. Many studies have focused on fine-tuning T5 [62] for this purpose [42, 44, 60, 63, 69]. Recently, the emergence of large language models (LLMs) such as GPT-4 [53, 56] and Gemini [2, 66] has once again transformed the text-to-SQL domain. By leveraging these powerful LLMs, researchers can decompose the complex text-to-SQL task into simpler sub-tasks, including schema linking, text-to-SQL generation, SQL refinement, and SQL selection [21, 22, 58, 59, 74]. Each sub-task can be managed by an LLM-based agent using prompt engineering or fine-tuning techniques. Unlike previous studies that design complex multi-agent frameworks, this paper focuses on improving the core capability of these frameworks—text-to-SQL generation—using large-scale synthetic data.

2.2 Data Augmentation for Text-to-SQL

To address the limited coverage of publicly available datasets, numerous studies have proposed data augmentation methods to generate additional training data (*i.e.*, <question, SQL query> pairs) that align with the distribution of existing datasets. Many approaches [25, 34, 43, 76, 80, 88, 91] employ a "SQL-to-question" pipeline, where SQL templates or grammars are used to generate SQL queries, which are then translated into natural language questions using neural network models. This approach ensures high-quality synthetic <question, SQL> pairs, as converting SQL to natural language is generally easier due to the flexibility of natural language. In these methods, SQL templates are often extracted from existing datasets [25, 28, 43, 91], while abstract syntax tree grammars typically require expert design [76, 80, 88]. However, SQL templates limit diversity, and grammar-based methods are labor-intensive, making both of them difficult to scale.

Alternatively, some studies adopt a "question-to-SQL" pipeline, where questions are first generated and then translated into SQL queries using off-the-shelf text-to-SQL models [83]. However, inaccuracies in used text-to-SQL models often result in noisy data. Other methods use question-to-SQL template pairs, either manually crafted or extracted from datasets, to generate new samples by filling slot mappings [79, 85, 86]. While effective, these approaches suffer from limited diversity and unnatural question generation.

The most closely related work is Sense [82], which employs GPT-4 to directly synthesize new data samples through carefully designed prompts. Specifically, Sense instructs GPT-4 to first generate a database, then formulate a question based on that database, and finally produce the corresponding SQL query for the question. However, Sense's reliance on expensive GPT-4 limits its scalability and cost-effectiveness. In contrast, our framework decouples the synthesis process into simpler, more controllable steps, enabling the use of less powerful yet open-source LLMs. As a result, our approach is highly cost-effective, particularly when scaling to million-level or larger synthesis requirements. Additionally, Sense's "question-to-SQL" design risks generating incorrect SQL queries for the previously synthesized questions, whereas our adopted "SQL-to-question" strategy ensures higher-quality synthetic data. Beyond these advantages, our method also synthesizes CoT solutions, further enhancing the interpretability of the generated data.

3 DATA SYNTHESIS FRAMEWORK

3.1 Overview

As illustrated in Figure 1, our data synthesis framework adopts a progressive pipeline comprising four key steps: web table-driven database synthesis, complexity-aware SQL query generation, stylized natural language question synthesis, and chain-of-thought solution synthesis. Each step leverages large language models (LLMs) in conjunction with automated pre-processing and post-processing strategies to ensure high-quality and diverse outputs, significantly reducing the reliance on extensive human intervention.

The process begins with web tables, which are abundant on websites and store structural data spanning a wide range of real-world domains. Using these tables, we synthesize databases that emulate realistic business scenarios. Next, based on synthetic databases, we generate SQL queries of varying complexity levels and back-translate them into natural language (NL) questions with diverse language styles. Finally, for each synthetic <database, question, SQL query> triplet, we produce a chain-of-thought (CoT) solution that outlines the step-by-step reasoning process used to derive the SQL query from the natural language question.

3.2 Web Table-Driven Database Synthesis

Developing robust text-to-SQL models requires fine-tuning on diverse databases. However, real-world databases are scarce on the internet because enterprise databases often contain sensitive information. Despite this, we observe that tabular data is abundant [5, 17, 31] and also reflects real-world scenarios for structured data storage. This wealth of tabular data presents a unique opportunity to address the aforementioned challenges. Leveraging this,

[1] https://github.com/RUCKBReasoning/OmniSQL

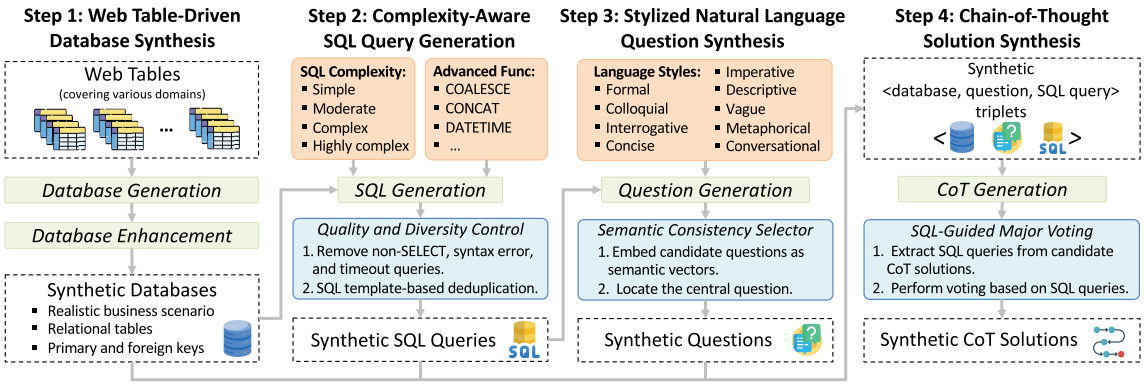


Figure 1: Illustration of the proposed text-to-SQL data synthesis framework.

we propose a new database synthesis method comprising two key steps: table-driven database generation and database enhancement.

Table-driven database generation. Specifically, given a web table, we prompt the LLM to first create a realistic business scenario hidden behind the given table and then design a relational database that could store the data relevant to this scenario. Each generated database includes several relational tables, along with structural information such as primary and foreign keys. Each relational table is defined by a table name, a description, column names, column data types, column descriptions, and two example data rows.

To encourage the synthesis of complex databases, we instruct the LLM to generate K relational tables. In this work, K is an integer sampled from a normal distribution $\mathcal{N}(10, 4^2)$. The prompt used for database generation comprises the following components: **(1) Task Instruction**: Directs the LLM to generate a business scenario and a corresponding database containing K relational tables, based on the given web table. **(2) 2-Shot Demonstrations**[2]: Includes two human-crafted examples of database synthesis, each presented as a <web table, business scenario, database> triplet. **(3) Web Table**: The input web table that serves as the seed for database generation.

Database Enhancement. Although the LLM can generate meaningful and realistic databases by following our instructions, we observe two practical issues with the synthetic databases: overly simplistic relational tables (averaging 4 columns per table) and incomplete primary and foreign key relationships. We attribute these limitations to shortcut learning in LLMs, where models may prioritize fulfilling user instructions with minimal effort rather than achieving optimal complexity or completeness [15]. To address these shortcomings and ensure the synthetic databases meet the complexity requirements of real-world applications, we introduce a database enhancement step following the initial generation.

In this step, given an initially generated database, we prompt the LLM to expand it by adding relevant columns to each relational table and completing any missing primary and foreign key relationships. This enhancement increases the average number of columns per table and improves the structural integrity of the databases. The prompt used for the database enhancement consists of the following components: **(1) Task Instruction**: Directs the LLM to enhance the structure of the given database. **(2) Database Information**: The initially generated database along with its business scenario.

Our database synthesis method is highly scalable, as each web table can be transformed into a database, and millions of web tables are readily available online [17]. Furthermore, previous work [82] has shown that LLMs are already proficient at generating high-quality database schemas, which can be attributed in part to the extensive presence of `CREATE TABLE` statements in large web-scale corpora. To empirically support this claim, we analyze the pre-training data used for StarCoder [46], a prominent code-focused LLM, and find 975,420 `.sql` files. Notably, 42% of these SQL files contain at least one `CREATE TABLE` statement. This widespread exposure to database schema definitions during pre-training likely contributes to the strong database generation capabilities observed in LLMs. In contrast, existing methods for collecting large-scale relational databases often struggle with scalability or quality issues. For example, GitSchemas [14] and SchemaPile [13] extract database construction statements (*e.g.*, `CREATE TABLE` and `ALTER TABLE`) from SQL files on GitHub. However, the availability of high-quality SQL files is limited compared to the abundance of web tables, restricting their scalability. WikiDBs [73] constructs databases using web tables from Wikidata. Specifically, it incrementally builds databases by starting with a single table and extending it with related tables. However, this approach risks connecting unrelated tables, potentially compromising database consistency.

3.3 Complexity-Aware SQL Query Generation

After generating databases, the next step is to synthesize SQL queries. In this paper, we leverage LLMs for SQL query generation, as their pre-training exposes them to a wide range of SQL scripts and snippets, making them well-suited for the task. However, early experiments revealed a challenge: smaller LLMs tend to generate overly simple SQL queries, while larger LLMs often produce highly complex ones, leading to an imbalance in query complexity. Recent literature [50] also reports a similar imbalance in query complexity when employing LLMs to generate SQL queries from the query descriptions of the TPC-DS [71] benchmark. To address this issue and control the complexity distribution of the synthetic data, we define four complexity levels—simple, moderate, complex, and

[2]Examples of the 2-shot demonstrations are available at: https://github.com/RUCKBReasoning/OmniSQL/blob/main/data_synthesis/database_synthesis/prompt_templates/schema_prompt.txt

highly complex—and instruct the LLM to generate SQL queries that align with a specified level. Additionally, we provide the LLM with advanced SQL functions and sampled database values to ensure the generated queries are meaningful and realistic. Furthermore, since many real-world natural language questions seek specific items (*e.g.*, a statistical number or a person's name), the corresponding SQL queries often return a limited number of columns. To reflect this, we impose a constraint on the number of returned columns during SQL query synthesis. Specifically, the prompt for SQL query synthesis includes the following components:

- **Task Instruction**: Directs the LLM to generate meaningful SQL queries that meet real-world data analysis needs.
- **Database Schema**: Includes the `CREATE TABLE` statements for all relational tables in the given database.
- **Advanced SQL Functions**: Randomly samples a few advanced SQL functions supported by the database engine, allowing the LLM to incorporate these functions in its queries when appropriate[3]. Each SQL function is presented with its name and a detailed description to help the LLM properly use it.
- **Database Values**: Randomly samples a few columns along with their stored values to assist the LLM in generating meaningful and contextually relevant predicates.
- **SQL Complexity**: Randomly samples a complexity level from ["Simple", "Moderate", "Complex", "Highly complex"]. Each level is defined by specific criteria along with an example SQL query.
- **Column Selection Constraint**: Specifies the number of columns the synthetic SQL query must select. This value is sampled from a geometric distribution with a success probability of $p = 0.6$. This distribution is chosen because it naturally biases the sampling toward smaller numbers, reflecting the common text-to-SQL scenario where SQL queries typically select a few columns.

In the post-processing stage, we apply several quality control measures. First, we filter out non-SELECT queries using pre-defined rules and execute the remaining queries on the synthetic databases to eliminate those with syntax errors or that result in timeouts. Then, to ensure diversity, we extract SQL templates[4] and retain only one query per template. For example, if the LLM generates "`SELECT name FROM school WHERE age > 18`" and "`SELECT name FROM school WHERE age > 55`", which share the same template, only one of these queries is retained in the final dataset.

3.4 Stylized NL Question Synthesis

After synthesizing SQL queries, the next step is to translate them into semantically equivalent natural language (NL) questions. Existing studies have primarily focused on ensuring the semantic accuracy of synthetic questions by introducing various novel techniques, such as hierarchical generation [80], intermediate representations [28], and pointer-decoder networks [91]. In this work, we argue that linguistic diversity is equally critical for developing robust text-to-SQL models, as real-world users express their questions in a wide range of styles. This is further supported by recent robustness benchmarks [8, 12, 19], which reveal that many text-to-SQL models struggle with linguistic perturbations, such as synonym substitution or sentence paraphrasing. Therefore, we advocate for a dual focus on both semantic accuracy and linguistic diversity during question synthesis.

To enhance linguistic diversity, we define nine language styles commonly observed in real-world user questions: formal, colloquial, imperative, interrogative, descriptive, concise, vague, metaphorical, and conversational. The first six styles (formal, colloquial, imperative, interrogative, descriptive, and concise) reflect scenarios where users express their intentions clearly but with variations in tone. In contrast, the vague and metaphorical styles represent cases where users employ ambiguous vocabulary or figurative language, often requiring external knowledge for interpretation. Finally, the conversational style simulates multi-turn dialogues, where users iteratively clarify their intentions. This style is particularly relevant in real-world applications, as users may not always express their needs directly, necessitating follow-up questions from the model. Examples illustrating these language styles can be found in our open-source GitHub repository[5].

We note that DBPal [79] also considers linguistic diversity during its question synthesis process. However, it relies on an off-the-shelf paraphrasing database [57] to replace synonyms in existing questions, which limits its ability to cover the full range of styles commonly used by users. Moreover, this simple synonym replacement mechanism may result in unnatural questions, further highlighting the need for a more robust approach to achieve linguistic diversity.

Specifically, the prompt for synthesizing natural language questions consists of the following components:

- **Task Instruction**: Directs the LLM to first generate an explanation of the provided SQL query and then translate it into a natural language question.
- **SQL Query**: The SQL query to be translated.
- **SQL-related Column Information**: Includes the names and descriptions of columns referenced in the SQL query. This aids the LLM in generating semantically accurate questions, particularly when column names are ambiguous, abbreviated, or coded.
- **Desired Language Style**: A randomly sampled style from the nine predefined styles, each accompanied by a description and an example question. For the formal, colloquial, imperative, interrogative, descriptive, and concise styles, the LLM generates stylized questions directly. For the vague and metaphorical styles, the LLM additionally provides the external knowledge underlying the question. For the conversational style, the LLM generates a multi-turn dialogue between <User> and <Assistant>.

For each synthetic SQL query, we generate multiple candidate questions using the LLM. To select the most semantically accurate question, we introduce a semantic consistency selector module, inspired by [68, 88]. Specifically, we utilize Sentence Transformers [67] to embed the candidate questions into vector representations[6]. For each candidate question, we compute its average cosine similarity with all other candidates. The question with the highest

[3]This paper focuses on the SQLite engine, as many text-to-SQL benchmarks are based on it. The functions supported by SQLite, including names and descriptions, can be found in the official documentation: https://www.sqlite.org/lang_corefunc.html.

[4]Templates are extracted by masking only the values in SQL queries. For example, given the SQL query "`SELECT name FROM school WHERE age > 18`", its template is "`SELECT name FROM school WHERE age > [MASK]`".

[5]https://github.com/RUCKBReasoning/OmniSQL/blob/main/assets/example_questions.png

[6]We use the "all-mpnet-base-v2" model for sentence embedding due to its superior embedding quality. Further details are available at https://sbert.net/docs/sentence_transformer/pretrained_models.html.

average similarity is selected, as it lies closest to the semantic center of the candidate set. By combining the pre-defined language styles with the semantic consistency selector, we enhance both the linguistic diversity and semantic accuracy of the synthetic questions.

3.5 Chain-of-Thought Solution Synthesis

Chain-of-thought (CoT) reasoning has demonstrated remarkable success across various challenging tasks [35, 78]. By decomposing complex problems into smaller, manageable steps, this approach enables LLMs to tackle intricate tasks more effectively, improving both accuracy and interpretability. Building on this, we augment the synthetic <database, question, SQL query> triplets by generating CoT solutions that explicitly outline the reasoning process behind constructing the SQL query from the question. The prompt for synthesizing CoT solutions consists of the following components:

- **Task Instruction**: Directs the LLM to generate a step-by-step CoT solution using the provided information.
- **Database Schema**: Includes the `CREATE TABLE` statements for all relational tables in the database.
- **NL Question and SQL Query Pair**: The natural language question and its corresponding SQL query.

A typical CoT solution begins by analyzing the question to identify the key information required. It then determines the relevant tables, columns, and filtering criteria needed to retrieve the desired data. Finally, it constructs the SQL query step by step, incorporating necessary joins, filters, aggregations, groupings, and other operators, culminating in the complete SQL query as the final answer.

Interestingly, in our preliminary experiments, we observe that the SQL queries generated by the synthetic CoT sometimes differ from the original ones. Upon closer examination, we find that CoT-generated SQL queries often better align with the questions compared to the original SQL queries. This improvement arises because the original <database, question, SQL query> triplets occasionally contain minor issues, such as unnecessary column selections, and incorrect join paths. The CoT synthesis process allows the LLM to identify and correct these issues during step-by-step reasoning, resulting in more accurate and refined SQL queries. This observation also aligns with prior research showing that LLMs excel at detecting and resolving minor errors in predicted SQL queries [22, 58, 70]. Thus, incorporating CoT not only provides detailed solutions but also enhances the overall quality of the synthetic data.

To enhance the diversity and reliability of synthetic CoT solutions, we generate multiple candidate CoT solutions for each synthetic <database, question, SQL query> triplet. To select the most reliable CoT solution, we extract SQL queries from these candidates and perform a majority vote. Specifically, we group candidates based on the execution results of their SQL queries. The final CoT solution is selected from the group with the most votes.

4 SYNSQL-2.5M: A MILLION-SCALE DATASET

To demonstrate the effectiveness of our data synthesis framework, we introduce SynSQL-2.5M —the first million-scale text-to-SQL dataset, entirely generated automatically by LLMs. SynSQL-2.5M contains 2,544,390 high-quality text-to-SQL samples, each represented as a <database, question, SQL query, CoT solution> quadruple. The dataset spans 16,583 synthetic databases across a wide range of real-world domains, including social media sentiment analysis, product inventory management, movie analytics, galaxy morphological analysis, and more.

To mitigate potential bias introduced by the habits of specific LLMs, we employ multiple LLMs during the data synthesis process, with each model responsible for generating a portion of the data. A key advantage of our framework is its decomposition of the text-to-SQL data synthesis task into four simple and manageable sub-tasks. Each sub-task can be effectively handled by relatively smaller, open-source LLMs, which are locally deployable and cost-efficient. This design eliminates the need for heavy reliance on expensive closed-source LLMs, as seen in prior studies [82]. In practice, we utilize models from multiple open-source model families, including Llama3.1 [16] (Meta-Llama-3.1-8B/70B-Instruct), Deepseek Coder [9, 26] (DeepSeek-Coder-6.7B/33B-Instruct, DeepSeek-Coder-V2-Lite-Instruct), Qwen2.5 [81] (Qwen2.5-7B/14B/32B/ 72B-Instruct), and Qwen2.5 Coder [30] (Qwen2.5-Coder-7B/14B/32B-Instruct). Larger models are assigned a greater share of the synthesis workload to ensure data quality.

To construct SynSQL-2.5M, we sample 0.1% of data from a tabular corpus, TabLib [17], resulting in approximately 1,319,561 web tables. Since these tables are sourced from websites, they often contain incomplete, redundant, or irrelevant content, which could compromise the quality of synthetic databases. To address this, we design a systematic filtering pipeline with the following steps: (1) Language Filtering: Non-English tables are removed to align with the English benchmarks. (2) Size Filtering: Tables with fewer than 5 columns or 5 rows are eliminated. (3) Deduplication: Similar tables are removed based on table headers. (4) Semantic Evaluation: Qwen2.5-72B-Instruct [81] is used to assess and filter out tables with insufficient semantic richness. After filtering, 19,935 high-quality web tables are retained. During the database synthesis process, 16,583 tables are successfully expanded into structurally complete databases, despite encountering parsing errors in LLM-generated responses and database creation failures. The initially generated databases contain an average of 9.15 tables per database and 4.86 columns per table. After enhancement, these averages increase to 10.15 tables and 7.3 columns, respectively, resulting in databases with real-world complexity. All databases are hosted and managed using SQLite. In the SQL synthesis phase, we generate 300 SQL queries for each synthetic database, producing approximately 5 million synthetic SQL queries. After applying quality and diversity controls, around 2.5 million queries are retained. For the natural language question and CoT synthesis phase, we sample 8 responses from LLMs for each input prompt, using a temperature of 0.8.

In this section, we provide a comprehensive statistical analysis of SynSQL-2.5M, highlighting its quality, diversity, and complexity through comparisons with three widely-used standard datasets (WikiSQL [92], Spider [87], and BIRD [45]) and two domain-specific datasets (ScienceBenchmark [88] and EHRSQL [38]). Additionally, we evaluate the quality of SynSQL-2.5M using the "LLM-as-a-judger" approach, further underscoring its high overall quality.

4.1 Overall Statistics

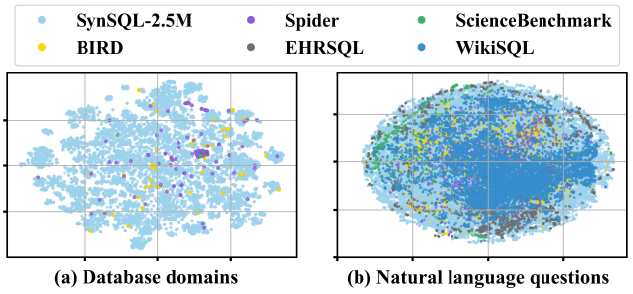
Table 1 provides a comparative overview of SynSQL-2.5M and other text-to-SQL datasets. As the statistics demonstrate, SynSQL-2.5M is

Table 1: Overall statistics of different datasets. Note: † denotes that each database in WikiSQL consists of a single table. ∗ indicates that the number of unique SQL queries for BIRD cannot be calculated due to the inaccessibility of its test set.

Dataset	Source	# Example	# Unique SQL	# DB	Lang. Styles	Knowledge	CoT Solution
WIKISQL [92]	Human+Template	80,654	80,257	26,531†	✘	✘	✘
Spider [87]	Human	10,181	4,489	200	✘	✘	✘
BIRD [45]	Human	12,751	-*	95	✘	✓	✘
ScienceBenchmark [88]	LLM-Gen+Human+Template	5,031	3,652	3	✘	✘	✘
EHRSQL [38]	Human+Template	20,108	18,253	2	✘	✘	✘
SynSQL-2.5M	LLM-Gen	2,544,390	2,412,915	16,583	✓	✓	✓

Table 2: Database statistics. The asterisk(*) means BIRD's database statistics are based on training and development sets due to the test set's inaccessibility. "DE" is the abbreviation of "database enhancement".

Dataset	# DB	# Tbl/DB	# Col/DB	# PK/DB	# FK/DB
WIKISQL [92]	26,531	1.00	6.34	1.00	0.00
Spider [87]	200	5.11	26.82	4.70	4.79
BIRD* [45]	95	7.64	54.56	6.71	6.58
ScienceBenchmark [88]	3	16.67	86.67	14.33	20.33
EHRSQL [38]	2	13.50	92.00	13.50	17.00
SynSQL-2.5M (w/o DE)	16,583	9.15	44.47	9.15	8.56
SynSQL-2.5M	16,583	10.15	74.07	10.14	9.61



(a) Database domains (b) Natural language questions

Figure 2: Database domain and question visualizations.

a large-scale, cross-domain text-to-SQL dataset that encompasses a diverse range of unique SQL queries and databases. Unlike previous datasets, which rely heavily on human annotators, SynSQL-2.5M is entirely generated by LLMs, showcasing its high scalability and cost efficiency. Furthermore, SynSQL-2.5M incorporates diverse linguistic styles in its questions. For vague or metaphorical questions, external knowledge is synthesized to clarify their intent. Notably, to the best of our knowledge, SynSQL-2.5M is the first large-scale dataset to provide step-by-step CoT solutions.

4.2 Database Statistics

Database Complexity. Table 2 compares the complexity of our synthetic databases and human-collected databases from existing benchmarks, focusing on the average number of tables, columns, primary keys, and foreign keys per database. The results show that the complexity of our synthetic databases surpasses that of the three widely-used standard datasets (*i.e.*, WikiSQL, Spider, and BIRD). This demonstrates that LLMs can reliably synthesize databases that closely mirror real-world complexity.

Domain Visualization. Database names (*e.g.*, grain_data_analysis and music_album_analytics_and_management) often reflect the domain of their underlying data. To evaluate the domain diversity of synthetic databases, we leverage Sentence Transformers [67] to embed database names into semantic vectors. In this vector space, databases from similar domains are positioned closer together, while those from dissimilar domains are farther apart. Figure 2 (a) presents a t-SNE [72] visualization of these embeddings, highlighting that our synthetic databases exhibit significantly broader domain coverage compared to existing human-collected databases.

4.3 SQL Statistics

Table 3 provides a detailed comparison of synthetic SQL queries and existing datasets across 11 dimensions, showcasing the complexity, sophisticated features, and diversity of our synthetic SQL queries.

To assess SQL complexity, we analyze the number of tables, joins, functions, and tokens per SQL query[7]. The results reveal that our synthetic SQL queries are more complex than those in human-annotated datasets. For instance, synthetic queries average 1.75 joins per query, compared to 0.94 in BIRD and 0.48 in Spider.

To evaluate the sophisticated features, we calculate the number of SQL queries involving aggregation functions, set operators, subqueries, window functions, and common table expressions (CTEs). The results demonstrate that SynSQL-2.5M provides strong coverage of these advanced features, supporting complex query patterns. Notably, approximately 40% of synthetic SQL queries utilize CTEs, which enhance the organization and readability of complex queries, making them easier for developers to understand and maintain.

Finally, we assess SQL diversity by counting the number of unique skeletons[8] and unique functions. The results highlight the high diversity of synthetic SQL queries, with over 2 million unique SQL skeletons. Additionally, synthetic queries cover 83 unique functions supported by the database engine.

4.4 Question Statistics

To evaluate the diversity of synthetic questions in SynSQL-2.5M, we employ Sentence Transformers to embed questions into semantic vectors and visualize these embeddings using t-SNE. For clarity,

[7] SQL queries are tokenized based on whitespace.

[8] Skeletons are extracted by masking all tables, columns, and values in SQL queries. For example, given the SQL query "`SELECT name FROM school WHERE age > 18`", its skeleton is "`SELECT [MASK] FROM [MASK] WHERE [MASK] > [MASK]`".

Table 3: SQL statistics. "Agg." and "Func." are abbreviations of aggregations and functions. The asterisk(*) means BIRD's SQL statistics are based on training and development sets due to test set inaccessibility.

Dataset	# Tables per SQL	# Joins per SQL	# Func. per SQL	# Tokens per SQL	# Agg.	# Set Operators	# Subqueries	# Window Func.	# CTEs	# Unique Skeletons	# Unique Func.
WIKISQL [92]	1.00	0	0.72	10.32	22,707	0	0	0	0	488	5
Spider [87]	1.52	0.48	0.51	14.85	3,802	348	604	0	0	2,136	5
BIRD* [45]	1.98	0.94	0.67	24.75	5,144	25	843	10	9	4,596	24
ScienceBenchmark [88]	1.42	0.42	0.25	14.61	1,234	2	23	0	0	855	5
EHRSQL [38]	3.60	0.62	**2.63**	41.62	11,673	166	17,989	3,207	0	2,447	11
SynSQL-2.5M	**4.00**	**1.75**	1.49	**57.05**	**1,897,440**	**363,472**	**612,647**	**33,916**	**1,073,483**	**2,190,988**	**83**

we randomly sample 0.5 million synthetic questions from SynSQL-2.5M for visualization. As illustrated in Figure 2 (b), SynSQL-2.5M demonstrates extensive coverage, effectively encompassing the distributions of human-annotated datasets.

4.5 Data Quality Evaluation

4.5.1 GPT-4o as Evaluators. We comprehensively evaluate SynSQL-2.5M using GPT-4o, a state-of-the-art LLM, following the paradigm of recent LLM-as-a-judge studies [23, 89, 93]. The evaluation encompasses four key aspects: **Database Aspect** (normalization compliance, field definition, relationships between tables, and schema complexity), **Question Aspect** (unambiguous phrasing, consistency with database schema, proper grammar, and real-world relevance), **SQL Query Aspect** (correctness, efficiency, maintainability, and security), and **Data Sample Aspect** (result alignment, structural alignment, efficiency of solution, and answer adherence). Detailed descriptions of each criterion can be found in our GitHub repository[9]. GPT-4o then assigns a rating (excellent, good, average, or poor) for every criterion on each data sample, along with detailed explanations. We aggregate the results using a weighted average:

$$Score = \frac{N_e \times 1.0 + N_g \times 0.75 + N_a \times 0.5 + N_p \times 0.25}{N_e + N_g + N_a + N_p}, \quad (1)$$

where N_e, N_g, N_a, and N_p are the numbers of samples rated as excellent, good, average, and poor. For comparison, we also conduct the same evaluation on BIRD [45], a widely used human-annotated benchmark. To ensure fairness, we randomly select 1,000 samples from each of SynSQL-2.5M and BIRD for evaluation. As shown in Figure 3, SynSQL-2.5M outperforms BIRD across nearly all criteria.

4.5.2 Human Experts as Evaluators. To further validate data quality, we conduct a human evaluation on 1,000 sampled data points. The evaluation is performed by three senior graduate students with backgrounds in computer science and active research interests in text-to-SQL and AI4DB. To reduce evaluator workload, we streamline the criteria: human experts only need to provide a binary judgment (pass/fail) for each of the following aspects—database realism and completeness, question meaningfulness, SQL query appropriateness, and overall correctness of the data sample.

To ensure diversity among the evaluated samples, we first randomly select 1,000 synthetic databases. For each database, we then randomly sample one complete text-to-SQL data instance, which includes a database schema, a natural language question, an SQL query, and its corresponding chain-of-thought (CoT) solution. This process results in 1,000 distinct samples for evaluation.

[9]https://github.com/RUCKBReasoning/OmniSQL/tree/main/GPT_evaluation/prompts

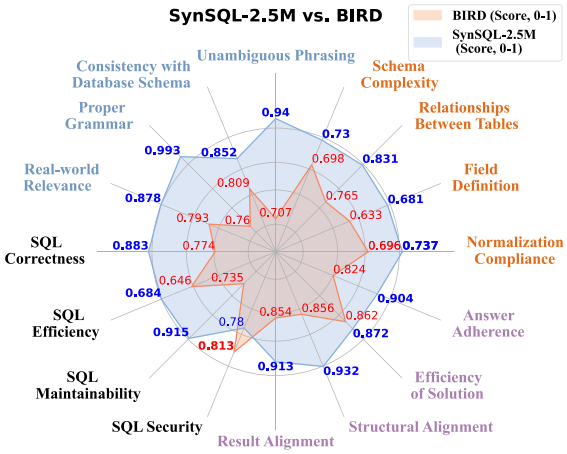


Figure 3: Quality evaluation of SynSQL-2.5M and BIRD judged by GPT-4o. Scores are computed using Equation 1.

Results show that 96% of synthetic databases are judged as realistic and structurally complete, making them suitable for practical use. Among the 1,000 complete text-to-SQL samples, 97% of questions are meaningful, 89% of SQL queries are appropriate, and 86% of the complete text-to-SQL data samples are fully correct. These findings indicate that the majority of our synthetic data is high-quality silver-standard data, suitable for model training.

5 OMNISQL: STATE-OF-THE-ART OPEN-SOURCE TEXT-TO-SQL LLM

Using our synthetic dataset, SynSQL-2.5M, along with two widely-adopted datasets, Spider and BIRD, we present OmniSQL, a powerful text-to-SQL model available in three scales: 7B, 14B, and 32B. In this section, we describe the construction of input-output sequence pairs for training OmniSQL and outline its training objectives. Additional implementation details are provided in Section 6.1.3.

5.1 Input-Output Construction

For all text-to-SQL data samples in the dataset, we first convert them into input-output sequence pairs to facilitate the training

of OmniSQL. Specifically, the input sequence consists of the database schema and the natural language question. Following previous studies [65, 82], the database schema is formatted as `CREATE TABLE` statements. Additionally, inspired by prior work [43, 70], we enrich the input with three supplementary elements: column descriptions, representative values, and question-relevant values, which are included as comments for each column.

Specifically, column descriptions assist the LLM in identifying the correct columns referenced in the question, particularly when column names are ambiguous (*e.g.*, abbreviations). Representative values for each column inform the LLM about the format of stored values, aiding in the use of advanced functions like `CONCAT`, `STRFTIME`, and `SUBSTR` in the generated SQL query. In practice, we select two distinct values for each column. Finally, following [43], we extract question-relevant values from the database, which can help the LLM to generate accurate predicates, especially when the question mentions specific database values.

The output sequence is the CoT solution, which includes step-by-step reasoning process and the final SQL query. However, since Spider and BIRD only provide gold SQL queries without CoT solutions, we enhance their training sets by synthesizing CoT solutions using the technique described in Section 3.5.

5.2 Supervised Fine-Tuning

We fine-tune the LLM using a conditional next token prediction loss, as described in [61]. The loss function is defined as follows:

$$\text{Loss} = -\mathbb{E}_{(x,y)\sim D}\left[\sum_t \log P_\theta(y_t \mid x, y_{<t})\right], \tag{2}$$

where D represents the training set, x and y are the input and output sequences, respectively, θ denotes the trainable parameters of the LLM, y_t is the t-th token in the output sequence, and $y_{<t}$ represents all preceding output tokens. OmniSQL is optimized to predict the CoT solution based on the provided database information and the corresponding question.

6 EXPERIMENTS

In this section, we comprehensively evaluate the performance of OmniSQL by comparing it with leading LLMs.

6.1 Experimental Setup

6.1.1 Evaluation Datasets. **Standard Benchmarks.** Spider [87] and BIRD [45] are widely used in prior text-to-SQL studies to evaluate the cross-domain text-to-SQL capabilities of models. Cross-domain refers to the fact that the databases in the different data splits (train/dev/test) have no overlap. For Spider, we use its dev and test sets, containing 1,034 and 2,147 data samples, respectively. For BIRD, due to the hidden nature of its test set, we only use its dev set, which consists of 1,534 data samples.

Challenging Domain-specific Benchmarks. Spider2.0 [39] is a new challenging benchmark focusing on the real-world enterprise text-to-SQL scenario. It contains highly complex SQL queries (often exceeding 100 lines), extremely long contexts (*e.g.*, large database schemas and massive external knowledge), and multiple SQL dialects (*e.g.*, BigQuery, Snowflake, SQLite, and DuckDB). Since SynSQL-2.5M is constructed using the SQLite dialect, we extract the SQLite portion of Spider2.0 for evaluation, containing 135 data samples. We name this subset as Spider2.0-SQLite[10]. Additionally, ScienceBenchmark [88][11] and EHRSQL [38][12] are two challenging benchmarks designed to evaluate text-to-SQL models in professional database domains. Specifically, ScienceBenchmark includes databases from the domains of research policymaking, astrophysics, and cancer research. EHRSQL, on the other hand, focuses on medical text-to-SQL applications. ScienceBenchmark and EHRSQL contain 299 and 1,008 evaluation samples, respectively. Since OmniSQL does not use any training data from these domain-specific benchmarks during fine-tuning, this evaluation can be treated as a zero-shot domain generalization scenario.

Robustness Benchmarks. Spider-DK [20], Spider-Syn [19], and Spider-Realistic [12] are three widely-adopted robustness benchmarks. Spider-DK tests the model's ability to understand implicit domain knowledge in natural language questions. Spider-Syn and Spider-Realistic modify questions in Spider's development set to replace explicit mentions of column names with their synonyms. These designs can mimic real-world users' questions, enabling the evaluation of model robustness in practical text-to-SQL applications. Specifically, Spider-DK, Spider-Syn, and Spider-Realistic offer 535, 1,034, and 508 samples for evaluation.

6.1.2 Evaluation Metrics. Following prior work, we use execution accuracy (EX) [87] and test-suite accuracy (TS) [90] as our evaluation metrics. EX measures whether the predicted SQL queries yield the same execution results as the gold SQL queries on a single database. TS extends EX's evaluation to multiple test-suite databases. Notably, only the Spider (dev), Spider-Syn, and Spider-Realistic datasets provide test-suite databases; therefore, these datasets are evaluated using TS, while the others are evaluated using EX. For both metrics, higher values indicate better performance.

6.1.3 Implementation Details. OmniSQL-7B/14B/32B is built on the Qwen2.5-Coder-7B/14B/32B-Instruct models, a series of advanced code language models pre-trained and instruction-tuned on 92 programming languages. Specifically, OmniSQL-7B and OmniSQL-14B are fully fine-tuned, while OmniSQL-32B is fine-tuned using a parameter efficient technique, low-rank adaptation (LoRA) [27], due to limited GPU computational resources. For LoRA, we set $r = 256$ and $\alpha = 512$, integrating adapters into the `q_proj`, `k_proj`, and `v_proj` layers of the model, while keeping the original weights unchanged. We use the AdamW optimizer [48] with parameters $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-8}$ to optimize the training objective. The peak learning rates are set to $2e^{-5}$, $4e^{-6}$ and $2e^{-4}$ for OmniSQL-7B, OmniSQL-14B, and OmniSQL-32B, respectively. We use a learning rate schedule with a linear warmup for the initial 5% of training, followed by cosine decay to 10% of the peak rate. In addition, the batch size, number of epochs, context length, weight decay, and gradient clipping are uniformly set to 512, 2, 8192, 0.1, and 1.0,

[10] Our proposed data synthesis method can generate data for most mainstream SQL dialects (*e.g.*, PostgreSQL, MySQL, SQL Server, BigQuery, etc.) because LLMs largely have seen these dialects in their pre-training corpora. We select SQLite as the default for this study because many classical text-to-SQL benchmarks use SQLite to host their databases, simplifying database deployment.

[11] The databases provided by ScienceBenchmark are originally hosted in PostgreSQL. We manually convert them to SQLite to facilitate evaluations.

[12] Many natural language questions in EHRSQL are unanswerable and several SQL queries return empty execution results. We remove these data samples from EHRSQL to ensure a consistent and reliable evaluation.

respectively. To optimize GPU memory usage during training, we leverage the DeepSpeed-Zero framework [64] with bfloat16 mixed precision. In practice, training OmniSQL-7B/14B/32B requires approximately 6, 12, and 20 days, respectively, on a single machine equipped with 8 NVIDIA A800-SXM4-80GB GPUs. We also provide LoRA versions of OmniSQL-7B and OmniSQL-14B (referred to as OmniSQL-7B-LoRA and OmniSQL-14B-LoRA), using the same LoRA hyperparameters as OmniSQL-32B. This enables a direct comparison between full and LoRA-based fine-tuning approaches.

6.1.4 Environments. All experiments are conducted on two GPU servers, each equipped with 8 NVIDIA A800-SXM4-80GB GPUs, an Intel(R) Xeon(R) Platinum 8336C CPU, and 2TB of RAM. For LLM training, we utilize PyTorch [3] 2.1.0 and DeepSpeed [64] 0.10.3, while vLLM [36] 0.6.3 is employed for LLM inference.

6.1.5 Baselines. We compare OmniSQL with a wide range of LLMs, including closed-source LLMs such as GPT-4o-mini-2024-07-18 [55], GPT-4o-2024-11-20 [56], and GPT-4-Turbo-2024-04-09 [54], as well as open-source LLMs like DeepSeek-V3 [10], DeepSeek Coder [9, 26], Qwen2.5 [81], Qwen2.5 Coder [30], LLama-3.1 [16], Granite [52], Mixtral [33], OpenCoder [29], and Starcoder2 [49]. These models span a wide range, from closed-source to open-source, small-scale (6.7B) to large-scale (671B), general-purpose to code-specific, and dense architectures to Mixture-of-Expert (MoE) designs. For closed-source LLMs with undisclosed parameter sizes, their performance serves as a point of reference rather than a direct comparison under controlled parameter size conditions. Additionally, due to the prohibitively high inference costs, reasoning models such as OpenAI o1 [32] and DeepSeek-R1 [11] are not considered.

6.1.6 Inference Strategy. During LLM inference, we explore greedy decoding and sampling. Greedy decoding, with a temperature of 0, ensures deterministic responses. Sampling, at a temperature of 0.8, introduces creativity and diversity, generating 8 candidate responses per sample. Then, we extract SQL queries from these candidates and perform majority voting based on execution results. The final response is selected from the group with the most votes.

6.2 Main Results

The evaluation results are shown in Table 4, where "Gre" and "Maj" denote greedy decoding and majority voting results, respectively. At each model scale, we compare OmniSQL with open-source LLMs of similar or larger size. Our findings are listed as follows:

Synthetic data significantly enhances the base model's text-to-SQL capabilities. A comparison between the Qwen2.5-Coder models and OmniSQL demonstrates that fine-tuning with SynSQL-2.5M leads to improved performance across most datasets. Specifically, under the greedy decoding strategy, OmniSQL-7B achieves an average improvement of +8.4% (from 52.8% to 61.2%) over its base model, Qwen2.5-Coder-7B-Instruct. Similarly, OmniSQL-14B and OmniSQL-32B show average improvements of 2.9% (from 59.3% to 62.2%) and 2.4% (from 60.8% to 63.2%) over their base models, Qwen2.5-Coder-14B-Instruct and Qwen2.5-Coder-32B-Instruct, respectively. Notably, OmniSQL-7B sets a new standard for 7B-scale LLMs in this domain. Furthermore, OmniSQL-14B and OmniSQL-32B represent the new state-of-the-art text-to-SQL capabilities.

OmniSQL consistently demonstrates leading performance on standard benchmarks, including Spider (dev), Spider (test), and BIRD (dev). Notably, OmniSQL-7B model attains an accuracy of 87.9% (greedy decoding) on Spider's test set, surpassing the best publicly available method on Spider's leaderboard[13], DAIL-SQL + GPT-4 + Self-Consistency (86.6%) [21], by 1.3%. Additionally, by employing a simple majority voting strategy, OmniSQL's performance on the Spider test set improves to 88.9%, 88.3%, and 89.8% for the 7B, 14B, and 32B model scales, respectively. In addition, we observe that increasing the model scale does not yield consistent performance gains on Spider. This is likely due to Spider being a relatively simple benchmark, where smaller-scale but powerful models already achieve near-optimal performance. In contrast, as BIRD is more challenging than Spider, we can observe slight performance improvements as the scale of OmniSQL increases. Notable, OmniSQL-32B achieves 67.0% (major voting) accuracy on BIRD's development set, making it competitive with Distillery + GPT-4o (67.2%) [51], which fine-tunes GPT-4o on BIRD's training set.

OmniSQL demonstrates exceptional domain generalization capabilities on domain-specific benchmarks, including Spider2.0-SQLite, EHRSQL, and ScienceBenchmark. On the most challenging benchmark, Spider2.0-SQLite, despite having limited model parameters, OmniSQL still shows comparable accuracy to much larger models. Notably, for 7B-scale baseline LLMs, their accuracy is only in the range from 0.7% to 3.7%, while OmniSQL-7B achieves 10.4% (greedy decoding) accuracy, underscoring the potential of smaller models to handle complex text-to-SQL problems. Then, on EHRSQL and ScienceBenchmark, OmniSQL consistently outperforms similar scale competitors and matches the performance of leading LLMs. Remarkably, on EHRSQL, which involves medical domain databases, OmniSQL-32B achieves the best performance, 46.8% (major voting), outperforming GPT-4o (45.5% in major voting) by 1.3%. These results underscore OmniSQL's capability to generalize effectively across professional domain databases without necessitating extensive domain-specific fine-tuning.

Interestingly, some open-source LLMs (*e.g.*, Qwen2.5-Coder-32B-Instruct and Meta-Llama-3.1-70B-Instruct) significantly outperform closed-source LLMs on standard datasets. However, their performance advantage diminishes on domain-specific datasets. This is likely because these open-source models incorporated the training sets of popular text-to-SQL benchmarks (*e.g.*, Spider and BIRD) during pre-training or fine-tuning. While this enables strong performance on familiar datasets, their effectiveness decreases when encountering out-of-domain datasets. In contrast, OmniSQL demonstrates consistently strong performance, excelling on both standard and domain-specific benchmarks.

OmniSQL demonstrates strong robustness on Spider-DK, Spider-Syn, and Spider-Realistic. On Spider-Syn and Spider-Realistic, OmniSQL outperforms baselines in most cases, demonstrating robustness to synonym substitution—crucial for real-world scenarios where users may use similar terms for tables or columns. However, on Spider-DK, we note that OmniSQL-14B and OmniSQL-32B underperform compared to their base models (*i.e.*, Qwen2.5 Coder) on Spider-DK. We attribute this to the fact that Spider-DK

[13] Spider's leaderboard can be found at https://yale-lily.github.io/spider. The top-ranked method, MiniSeek, is excluded from comparisons as it is not publicly available.

Table 4: Main results on 9 datasets (%). The best results are highlighted in bold. "DSC" is the abbreviation of "DeepSeek-Coder".

LLM	Spider (dev)		Spider (test)		BIRD (dev)		Spider2.0-SQLite		Science Benchmark		EHRSQL		Spider-DK		Spider-Syn		Spider-Realistic		Average	
	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj
Closed-source LLMs (as a reference)																				
GPT-4o-mini	70.4	71.0	82.4	83.7	58.8	61.5	5.9	7.4	51.8	52.5	37.9	43.1	73.3	74.4	60.5	61.6	64.4	66.7	56.2	58.0
GPT-4-Turbo	72.4	72.2	83.4	84.2	62.0	63.6	11.9	11.9	59.2	59.5	43.1	44.8	72.3	72.1	62.9	63.5	67.5	68.3	59.4	60.0
GPT-4o	70.9	70.7	83.2	84.9	61.9	64.0	14.8	15.6	55.5	56.2	44.9	45.5	72.9	73.5	59.6	62.3	66.5	66.7	58.9	59.9
Open-source LLMs (~7B)																				
DSC-6.7B-Instruct	63.2	63.2	70.5	73.2	43.1	48.0	3.0	3.7	40.8	45.5	28.6	33.9	60.9	64.1	49.9	51.7	58.7	58.9	46.5	49.1
Qwen2.5-Coder-7B-Instruct	73.4	77.1	82.2	85.6	50.9	61.3	1.5	2.2	45.2	51.2	24.3	36.9	67.5	73.6	63.1	66.9	66.7	70.5	52.8	58.4
Qwen2.5-7B-Instruct	65.4	68.9	76.8	82.6	46.9	56.4	1.5	1.5	38.5	47.5	20.9	32.1	63.7	71.8	54.2	60.0	56.7	63.6	47.2	53.8
OpenCoder-8B-Instruct	59.5	59.5	68.3	70.1	37.5	45.3	0.7	0.7	39.8	45.5	21.9	29.9	62.6	64.7	46.0	46.1	49.0	49.4	42.8	45.7
Meta-Llama-3.1-8B-Instruct	61.8	67.7	72.2	78.5	42.0	53.1	3.0	2.2	36.8	43.1	24.6	33.7	62.6	69.9	53.1	59.3	57.5	61.0	46.0	52.1
Granite-8B-Code-Instruct	58.5	59.2	64.9	68.6	27.6	32.5	0.7	0.7	29.4	31.4	16.0	22.6	50.7	54.4	45.0	46.8	48.8	49.4	38.0	40.6
Granite-3.1-8B-Instruct	58.3	65.0	69.8	75.3	36.0	47.2	1.5	1.5	36.8	47.5	19.6	32.3	60.0	66.5	47.7	53.8	46.5	57.1	41.8	49.6
OmniSQL-7B-LoRA	79.9	79.3	85.6	87.1	61.5	**66.6**	8.9	**10.4**	**50.2**	53.8	**39.4**	**45.6**	74.2	76.6	66.2	68.6	76.0	75.2	60.2	62.6
OmniSQL-7B	**81.2**	**81.6**	**87.9**	**88.9**	**63.9**	66.1	**10.4**	**10.4**	**50.2**	**55.9**	34.9	40.0	**76.1**	**77.8**	**69.7**	**69.6**	**76.2**	**78.0**	**61.2**	**63.1**
Open-source LLMs (14B-32B)																				
Qwen2.5-Coder-14B-Instruct	78.1	80.6	86.6	88.0	61.5	**66.1**	5.9	4.4	52.2	54.2	31.6	35.5	73.6	**77.8**	68.2	69.3	76.2	74.2	59.3	61.1
Qwen2.5-14B-Instruct	66.5	69.7	82.0	84.0	56.7	62.1	5.2	10.4	51.2	56.2	28.8	35.2	72.3	74.0	58.1	60.7	62.4	65.2	53.7	57.5
Starcoder2-15B-Instruct	65.8	67.6	73.0	74.0	38.5	42.6	0.7	3.0	25.8	29.8	16.8	22.6	66.5	68.2	49.4	52.4	56.7	61.0	43.7	46.8
DSC-V2-Lite-In. (16B, MoE)	68.0	70.0	77.9	79.6	44.6	51.8	3.0	5.2	39.1	45.8	23.9	32.4	63.7	67.5	55.6	57.9	61.8	64.0	48.6	52.7
Granite-20B-Code-Instruct	65.7	63.6	74.1	72.9	34.0	40.5	0.0	0.0	37.5	40.1	23.5	26.9	62.2	63.9	52.3	54.3	55.7	56.3	45.0	46.5
Codestral-22B	66.7	67.5	78.6	81.0	52.7	56.8	5.2	5.9	48.5	54.2	37.8	40.4	69.9	72.7	55.2	59.4	62.6	64.8	53.0	55.9
OmniSQL-14B-LoRA	80.9	80.9	88.0	87.8	63.6	65.6	**11.1**	11.9	**58.5**	55.2	38.6	**44.4**	**76.4**	76.8	**70.7**	**72.1**	**77.0**	**79.1**	**62.8**	63.8
OmniSQL-14B	**81.4**	**82.0**	**88.3**	**88.3**	**64.2**	65.9	10.4	**13.3**	56.9	**56.9**	**39.9**	43.6	72.9	74.8	69.0	72.0	76.4	78.5	62.2	**63.9**
Open-source LLMs (≥ 32B)																				
Qwen2.5-Coder-32B-Instruct	77.7	77.9	87.5	88.0	64.5	67.0	5.9	11.9	54.8	56.5	36.4	43.3	**78.3**	**78.1**	**69.9**	70.5	72.4	74.8	60.8	63.1
Qwen2.5-32B-Instruct	71.9	73.6	84.9	86.1	62.0	64.7	7.4	8.9	50.5	54.5	33.6	41.4	73.1	76.1	64.0	66.0	66.5	68.1	57.1	59.9
DSC-33B-Instruct	66.0	68.5	74.3	76.5	49.2	55.9	7.4	4.4	44.5	52.2	31.4	35.4	69.0	71.4	53.5	57.4	59.1	63.2	50.5	53.9
Granite-34B-Code-Instruct	69.9	70.0	74.4	77.0	33.8	41.3	0.7	1.5	40.1	40.1	23.8	29.9	64.7	70.7	55.6	59.8	60.0	59.6	47.0	50.0
Mixtral-8x7B-In. (47B, MoE)	54.4	59.0	67.8	74.1	35.3	42.9	2.2	2.2	29.4	34.8	21.5	31.4	55.3	60.4	42.1	48.8	48.0	53.3	39.6	45.2
Meta-Llama-3.1-70B-Instruct	72.3	71.0	84.3	85.9	**65.1**	**67.4**	3.0	3.7	55.2	56.2	37.4	41.4	75.1	**78.1**	61.7	63.1	64.0	65.6	57.6	59.2
Qwen2.5-72B-Instruct	73.9	72.1	84.0	85.7	60.3	63.6	9.6	14.8	52.8	58.2	35.0	41.2	76.4	77.6	64.1	64.3	70.1	68.5	58.5	60.7
DeepSeek-V3 (671B, MoE)	73.1	73.5	85.5	85.8	63.2	63.8	**12.6**	**15.6**	56.2	57.9	**43.2**	43.5	72.9	73.8	64.4	65.1	67.9	66.9	59.9	60.7
OmniSQL-32B	**80.9**	**80.9**	**87.6**	**89.8**	64.5	67.0	11.9	13.3	**57.2**	**58.5**	42.4	**46.8**	76.1	77.6	69.7	**72.1**	**78.1**	**77.2**	**63.2**	**64.8**

Table 5: Results of ablation studies (%). All scores are reported under greedy decoding. FT means "fine-tuning".

	Spider (dev)	Spider (test)	BIRD (dev)	Spider2.0-SQLite	Science Benchmark	EHRSQL	Spider-DK	Spider-Syn	Spider-Realistic
Qwen2.5-Coder-7B-Instruct (base model)	73.4	82.2	50.9	1.5	45.2	24.3	67.5	63.1	66.7
FT w/ SynSQL-2.5M	74.6	83.5	59.9	9.6	48.5	**37.2**	71.6	61.6	70.3
FT w/ CoT-enhanced Spider + BIRD	80.3	86.6	59.6	5.9	49.5	26.0	72.0	**70.0**	**76.4**
FT w/ original Spider + BIRD	76.9	80.7	55.1	3.0	43.8	31.4	65.8	65.9	71.9
FT w/ SynSQL-2.5M + CoT-enhanced Spider + BIRD (**OmniSQL-7B**)	**81.2**	**87.9**	**63.9**	**10.4**	**50.2**	34.9	**76.1**	69.7	76.2

requires the text-to-SQL model to have a deep understanding of implicit knowledge (*e.g.*, commonsense knowledge) hidden in the questions. The base models, trained on extensive corpora, likely possess a stronger foundation in such knowledge, enabling them to perform better on Spider-DK. These findings provide valuable insights for further refining our data synthesis framework. For example, we could introduce a new language style to guide the LLM in synthesizing questions that incorporate commonsense knowledge.

Smaller models benefit more from full fine-tuning, while LoRA is sufficient for larger models. For the 7B model, full fine-tuning (OmniSQL-7B) outperforms the LoRA-based version (OmniSQL-7B-LoRA) in most cases. However, this advantage becomes less pronounced with larger models (see OmniSQL-14B vs. OmniSQL-14B-LoRA). We hypothesize that larger models possess stronger inherent capabilities, so updating only a subset of parameters with LoRA is sufficient for effective adaptation and may also help mitigate overfitting compared to full fine-tuning.

Finally, it is important to note that OmniSQL's performance can be further enhanced by incorporating additional text-to-SQL techniques, such as question rephrasing [41], schema linking [42, 59], SQL revision [22, 70], and SQL selection [37, 58]. Recent open-source text-to-SQL frameworks like CHASE-SQL [58] and Alpha-SQL [41] leverage multi-stage pipelines and integrate a range of such techniques. In contrast, our work isolates and evaluates the core text-to-SQL capability of a single LLM in a single-step inference setup. To ensure fair evaluation, we focus on this setting and do not compare OmniSQL directly with multi-stage frameworks.

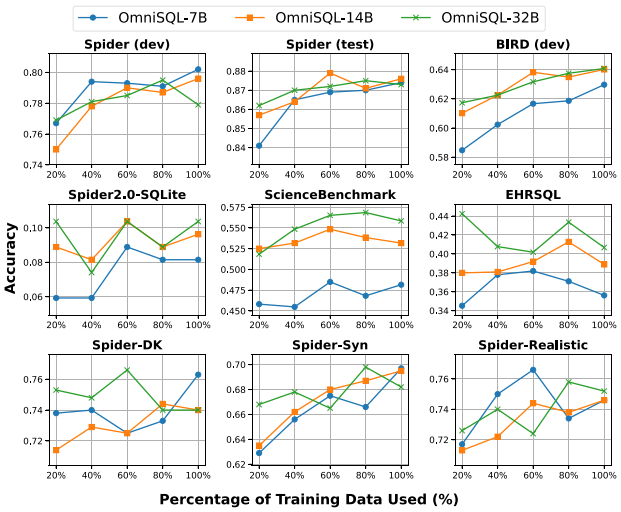


Figure 4: Performance changes (measured by accuracy; higher is better) with increasing training data.

6.3 Ablation studies

6.3.1 Ablations on Synthetic Data. To assess the impact of our synthetic data, we conduct two ablation studies. First, we fine-tune the base model using only SynSQL-2.5M. As shown in Table 5 (FT w/ SynSQL-2.5M), this approach yields substantial improvements across eight datasets compared to the base model, including notable gains of 9.0% on the BIRD dev set and 12.9% on EHRSQL, underscoring the effectiveness of SynSQL-2.5M. Second, to further demonstrate the contribution of SynSQL-2.5M, we exclude it from the training set of OmniSQL and fine-tune the base model solely on CoT-enhanced Spider and BIRD. By comparing "FT w/ CoT-enhanced Spider + BIRD" with "FT w/ SynSQL-2.5M + CoT-enhanced Spider + BIRD" in Table 5, we observe consistent performance drops across seven datasets when SynSQL-2.5M is omitted, highlighting its key role in complementing gaps left by human-annotated data.

6.3.2 Ablations on CoT Synthesis. We conduct ablation studies to assess the impact of synthesizing CoT solutions (the fourth step in our pipeline). Specifically, we fine-tune Qwen2.5-Coder-7B-Instruct on the original training sets of Spider and BIRD, which contain only gold SQL queries as labels. As shown in Table 5, removing CoT solutions from the training sets leads to notable performance drops on nearly all datasets (except EHRSQL), as seen by comparing the rows "FT w/ CoT-enhanced Spider + BIRD" and "FT w/ original Spider + BIRD". The improvements from CoT can be attributed to both enhanced reasoning in the LLM and the correction of mislabeled samples during CoT synthesis, thus improving data quality.

6.3.3 Ablations on Training Data Scale. We assess the effect of training data scale by fine-tuning models on randomly sampled subsets (20%, 40%, 60%, 80%, and 100%) for one epoch and reporting greedy decoding results in Figure 4. The results show that model performance consistently improves with larger training data. Notably, the upward trend persists even with the full dataset, suggesting further gains are possible with more data.

Table 6: Results of data augmentation methods (%). † denotes methods using the EX metric on Spider (dev). Columns show results without synthetic data (w/o), with synthetic data (w/), and the improvement (Δ) from synthetic data.

Method	Spider (dev)			BIRD (dev)		
	w/o	w/	Δ	w/o	w/	Δ
DT-Fixup+Syn data† [83]	74.6	76.1	+1.5	-	-	-
T5-3B+PICARD+Syn data† [28]	79.3	81.4	+2.1	-	-	-
Sense-13B [82]	79.4	82.9	+3.5	51.6	52.9	+1.3
OmniSQL-7B (greedy decoding)	76.9	81.2	**+4.3**	55.1	63.9	**+8.8**

6.3.4 Ablations on CoT Synthesis Stage. As discussed in Section 3.5, we empirically observe that adding the CoT synthesis step helps correct prior errors and improves SQL quality. To validate this, we sampled 5,000 cases from SynSQL-2.5M where SQL execution results differed before and after CoT synthesis. GPT-4o is then used to judge which SQL query better matches the question. In 4,699 out of 5,000 cases (93.98%), the SQL after CoT synthesis is preferred, confirming that this step substantially enhances the quality and reliability of the synthetic data.

6.4 Comparison with Data Augmentation

Finally, we compare our data synthesis method with 3 recent and representative data augmentation methods. To ensure a fair comparison, we focus on the performance improvements brought by synthetic data rather than absolute accuracy, as different methods use varying base models. As shown in Table 6, our method achieves significant accuracy gains on both the Spider and BIRD development sets, outperforming existing data augmentation methods by a large margin. Notably, while Sense [82] also uses LLMs for data synthesis, it achieves only a modest improvement (+1.3%) on BIRD. In contrast, our approach achieves a substantial +8.8% improvement on BIRD, highlighting its effectiveness.

7 CONCLUSION

This paper introduces a novel framework for text-to-SQL data synthesis, which breaks the process into four sequential, controllable steps. Using this framework, we introduce SynSQL-2.5M, a new text-to-SQL dataset containing 2.5 million high-quality samples. Comprehensive statistical analysis and quality evaluation demonstrate the superior quality, diversity, and complexity of SynSQL-2.5M. Based on this dataset, we introduce a new text-to-SQL model, OmniSQL. Extensive evaluations across nine benchmarks show that OmniSQL substantially outperforms baseline LLMs of similar scale and matches the performance of state-of-the-art models, despite using far fewer parameters. We believe that releasing both SynSQL-2.5M and OmniSQL will provide valuable resources and drive further advances in text-to-SQL research.

ACKNOWLEDGMENTS

This work is supported by the National Key Research & Development Plan of China (2023YFF0725100) and the National Natural Science Foundation of China (62322214, U23A20299, U24B20144, 62172424, 62276270). We also acknowledge the support of the Public Computing Cloud, Renmin University of China.

REFERENCES

[1] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases - an introduction. *Nat. Lang. Eng.* 1, 1 (1995), 29–81. https://doi.org/10.1017/S135132490000005X

[2] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, and et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *CoRR* abs/2312.11805 (2023). https://doi.org/10.48550/ARXIV.2312.11805 arXiv:2312.11805

[3] Jason Ansel, Edward Z. Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, and et al. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*. ACM, 929–947. https://doi.org/10.1145/3620665.3640366

[4] Abhijeet Awasthi, Ashutosh Sathe, and Sunita Sarawagi. 2022. Diverse Parallel Data Synthesis for Cross-Database Adaptation of Text-to-SQL Parsers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*. Association for Computational Linguistics, 11548–11562.

[5] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I (Lecture Notes in Computer Science)*, Vol. 9366. Springer, 425–441. https://doi.org/10.1007/978-3-319-25007-6_25

[6] Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. SADGA: Structure-Aware Dual Graph Aggregation Network for Text-to-SQL. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. 7664–7676.

[7] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. 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*. Association for Computational Linguistics, 2541–2555. https://doi.org/10.18653/V1/2021.ACL-LONG.198

[8] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, and et al. 2023. Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[9] DeepSeek-AI. 2024. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence. *CoRR* abs/2406.11931 (2024). https://doi.org/10.48550/ARXIV.2406.11931 arXiv:2406.11931

[10] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. *CoRR* abs/2412.19437 (2024). https://doi.org/10.48550/ARXIV.2412.19437 arXiv:2412.19437

[11] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL]

[12] 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*. Association for Computational Linguistics, 1337–1350. https://doi.org/10.18653/V1/2021.NAACL-MAIN.105

[13] Till Döhmen, Radu Geacu, Madelon Hulsebos, and Sebastian Schelter. 2024. SchemaPile: A Large Collection of Relational Database Schemas. *Proc. ACM Manag. Data* 2, 3 (2024), 172.

[14] Till Döhmen, Madelon Hulsebos, Christian Beecks, and Sebastian Schelter. 2022. GitSchemas: A Dataset for Automating Relational Data Preparation Tasks. In *38th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2022, Kuala Lumpur, Malaysia, May 9, 2022*. IEEE, 74–78.

[15] Mengnan Du, Fengxiang He, Na Zou, Dacheng Tao, and Xia Hu. 2022. Shortcut Learning of Large Language Models in Natural Language Understanding: A Survey. *CoRR* abs/2208.11857 (2022). https://doi.org/10.48550/ARXIV.2208.11857 arXiv:2208.11857

[16] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, and et al. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024). https://doi.org/10.48550/ARXIV.2407.21783 arXiv:2407.21783

[17] Gus Eggert, Kevin Huo, Mike Biven, and Justin Waugh. 2023. TabLib: A Dataset of 627M Tables with Context. *CoRR* abs/2310.07875 (2023). https://doi.org/10.48550/ARXIV.2310.07875 arXiv:2310.07875

[18] Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. 2023. CatSQL: Towards Real World Natural Language to SQL Applications. *Proc. VLDB Endow.* 16, 6 (2023), 1534–1547. https://doi.org/10.14778/3583140.3583165

[19] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards Robustness of Text-to-SQL Models against Synonym Substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*. Association for Computational Linguistics, 2505–2515. https://doi.org/10.18653/V1/2021.ACL-LONG.195

[20] Yujian Gan, Xinyun Chen, and Matthew Purver. 2021. Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Association for Computational Linguistics, 8926–8931. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.702

[21] 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://doi.org/10.14778/3641204.3641221

[22] 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. XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. *CoRR* abs/2411.08599 (2024). https://doi.org/10.48550/ARXIV.2411.08599 arXiv:2411.08599

[23] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Yuanzhuo Wang, and Jian Guo. 2024. A Survey on LLM-as-a-Judge. *CoRR* abs/2411.15594 (2024). https://doi.org/10.48550/ARXIV.2411.15594 arXiv:2411.15594

[24] 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. https://doi.org/10.1145/3589292

[25] Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question Generation from SQL Queries Improves Neural Semantic Parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 1597–1607. https://doi.org/10.18653/V1/D18-1188

[26] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, and et al. 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

[27] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[28] 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, and et al. 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*. Association for Computational Linguistics, 1327–1343. https://doi.org/10.18653/V1/2023.FINDINGS-ACL.86

[29] Siming Huang, Tianhao Cheng, J. K. Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, and et al. 2024. OpenCoder: The Open Cookbook for Top-Tier Code Large Language Models. *CoRR* abs/2411.04905 (2024). https://doi.org/10.48550/ARXIV.2411.04905 arXiv:2411.04905

[30] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, and et al. 2024. Qwen2.5-Coder Technical Report. *CoRR* abs/2409.12186 (2024). https://doi.org/10.48550/ARXIV.2409.12186 arXiv:2409.12186

[31] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proc. ACM Manag. Data* 1, 1 (2023), 30:1–30:17. https://doi.org/10.1145/3588710

[32] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, and et al. 2024. OpenAI o1 System Card. *CoRR* abs/2412.16720 (2024). https://doi.org/10.48550/ARXIV.2412.16720 arXiv:2412.16720

[33] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, and et al. 2024. Mixtral of Experts. *CoRR* abs/2401.04088 (2024). https://doi.org/10.48550/ARXIV.2401.04088 arXiv:2401.04088

[34] Hideo Kobayashi, Wuwei Lan, Peng Shi, Shuaichen Chang, Jiang Guo, Henghui Zhu, Zhiguo Wang, and Patrick Ng. 2025. You Only Read Once (YORO): Learning to Internalize Database Knowledge for Text-to-SQL. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long*

Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025. Association for Computational Linguistics, 1889–1901.

[35] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

[36] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*. ACM, 611–626. https://doi.org/10.1145/3600006.3613165

[37] 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, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*. Association for Computational Linguistics, 337–353.

[38] Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. 2022. EHRSQL: A Practical Text-to-SQL Benchmark for Electronic Health Records. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

[39] 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 I. Wang, and Tao Yu. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. *CoRR* abs/2411.07763 (2024). https://doi.org/10.48550/ARXIV.2411.07763 arXiv:2411.07763

[40] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? [Experiment, Analysis & Benchmark]. *Proc. VLDB Endow.* 17, 11 (2024), 3318–3331.

[41] Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025. Alpha-SQL: Zero-Shot Text-to-SQL using Monte Carlo Tree Search. *CoRR* abs/2502.17248 (2025).

[42] 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*. AAAI Press, 13067–13075. https://doi.org/10.1609/AAAI.V37I11.26535

[43] 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 (2024), 127. https://doi.org/10.1145/3654930

[44] 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*. AAAI Press, 13076–13084. https://doi.org/10.1609/AAAI.V37I11.26536

[45] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, and et al. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

[46] 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, and et al. 2023. StarCoder: may the source be with you! *Trans. Mach. Learn. Res.* 2023 (2023).

[47] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *CoRR* abs/2408.05109 (2024).

[48] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[49] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, and et al. 2024. StarCoder 2 and The Stack v2: The Next Generation. *CoRR* abs/2402.19173 (2024). https://doi.org/10.48550/ARXIV.2402.19173 arXiv:2402.19173

[50] Limin Ma, Ken Pu, Ying Zhu, and Wesley Taylor. 2025. Comparing Large Language Models for Generating Complex Queries. *Journal of Computer and Communications* 13 (2025), 236–249.

[51] 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. *CoRR* abs/2408.07702 (2024). https://doi.org/10.48550/ARXIV.2408.07702 arXiv:2408.07702

[52] Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, Manish Sethi, Xuan-Hong Dang, and et al. 2024. Granite Code Models: A Family of Open Foundation Models for Code Intelligence. *CoRR* abs/2405.04324 (2024). https://doi.org/10.48550/ARXIV.2405.04324 arXiv:2405.04324

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

[54] OpenAI. 2024. GPT-4 Turbo and GPT-4. (2024). https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4.

[55] OpenAI. 2024. GPT-4o mini: advancing cost-efficient intelligence. (2024). https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[56] OpenAI. 2024. Hello GPT-4o. (2024). https://openai.com/index/hello-gpt-4o/.

[57] Ellie Pavlick and Chris Callison-Burch. 2016. Simple PPDB: A Paraphrase Database for Simplification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. The Association for Computer Linguistics.

[58] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Ö. Arik. 2024. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. *CoRR* abs/2410.01943 (2024). https://doi.org/10.48550/ARXIV.2410.01943 arXiv:2410.01943

[59] 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*.

[60] 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*. Association for Computational Linguistics, 3215–3229. https://doi.org/10.18653/V1/2022.EMNLP-MAIN.211

[61] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[62] 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. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67.

[63] 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*. Association for Computational Linguistics, 150–160. https://doi.org/10.18653/V1/2023.ACL-SHORT.15

[64] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*. IEEE/ACM, 20. https://doi.org/10.1109/SC41405.2020.00024

[65] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the Text-to-SQL Capabilities of Large Language Models. *CoRR* abs/2204.00498 (2022). https://doi.org/10.48550/ARXIV.2204.00498 arXiv:2204.00498

[66] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy P. Lillicrap, Jean-Baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, Ioannis Antonoglou, Rohan Anil, Sebastian Borgeaud, Andrew M. Dai, Katie Millican, Ethan Dyer, Mia Glaese, Thibault Sottiaux, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, and et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *CoRR* abs/2403.05530 (2024). https://doi.org/10.48550/ARXIV.2403.05530 arXiv:2403.05530

[67] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084

[68] Gaetano Rossiello, Pierpaolo Basile, and Giovanni Semeraro. 2017. Centroid-based Text Summarization through Compositionality of Word Embeddings. In *Proceedings of the Workshop on Summarization and Summary Evaluation Across Source Types and Genres, MultiLing@EACL 2017, Valencia, Spain, April 3, 2017*. Association for Computational Linguistics, 12–21. https://doi.org/10.18653/V1/W17-1003

[69] 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. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.779

[70] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *CoRR* abs/2405.16755 (2024). https://doi.org/10.48550/ARXIV.2405.16755 arXiv:2405.16755

[71] Transaction Processing Performance Council (TPC). [n.d.]. TPC-DS: Decision Support Benchmark. Online. Available: http://www.tpc.org/tpcds/.

[72] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[73] Liane Vogel, Jan-Micha Bodensohn, and Carsten Binnig. 2024. WikiDBs: A Large-Scale Corpus Of Relational Databases From Wikidata. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

[74] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, 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, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*. Association for Computational Linguistics, 540–557.

[75] 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, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 7567–7578. https://doi.org/10.18653/V1/2020.ACL-MAIN.677

[76] Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021. Learning to Synthesize Data for Semantic Parsing. 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*. Association for Computational Linguistics, 2760–2766. https://doi.org/10.18653/V1/2021.NAACL-MAIN.220

[77] Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust Text-to-SQL Generation with Execution-Guided Decoding. *arXiv e-prints*, Article arXiv:1807.03100 (July 2018), arXiv:1807.03100 pages. https://doi.org/10.48550/arXiv.1807.03100 arXiv:1807.03100 [cs.CL]

[78] 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 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

[79] Nathaniel Weir, Prasetya Ajie Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Çetintemel, and Carsten Binnig. 2020. DBPal: A Fully Pluggable NL2SQL Training Pipeline. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2347–2361.

[80] Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data Augmentation with Hierarchical SQL-to-Question Generation for Cross-domain Text-to-SQL Parsing. 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*. Association for Computational Linguistics, 8974–8983. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.707

[81] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, and et al. 2024. Qwen2.5 Technical Report. *CoRR* abs/2412.15115 (2024). https://doi.org/10.48550/ARXIV.2412.15115 arXiv:2412.15115

[82] Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing Text-to-SQL Data from Weak and Strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 7864–7875. https://doi.org/10.18653/V1/2024.ACL-LONG.425

[83] Wei Yang, Peng Xu, and Yanshuai Cao. 2021. Hierarchical Neural Data Synthesis for Semantic Parsing. *CoRR* abs/2112.02212 (2021). arXiv:2112.02212

[84] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 8413–8426. https://doi.org/10.18653/V1/2020.ACL-MAIN.745

[85] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[86] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. 2018. SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 1653–1663. https://doi.org/10.18653/V1/D18-1193

[87] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and et al. 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, Brussels, Belgium, October 31 - November 4, 2018*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 3911–3921. https://doi.org/10.18653/V1/D18-1425

[88] 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. https://doi.org/10.14778/3636218.3636225

[89] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. 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*.

[90] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Association for Computational Linguistics, 396–411. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.29

[91] Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded Adaptation for Zero-shot Executable Semantic Parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Association for Computational Linguistics, 6869–6882. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.558

[92] 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

[93] Lianghui Zhu, Xinggang Wang, and Xinlong Wang. 2023. JudgeLM: Fine-tuned Large Language Models are Scalable Judges. *CoRR* abs/2310.17631 (2023). https://doi.org/10.48550/ARXIV.2310.17631 arXiv:2310.17631