arXiv:2511.10192v1 [cs.CL] 13 Nov 2025

TEXT2SQL-FLOW: A Robust SQL-Aware Data Augmentation Framework for Text-to-SQL

Qifeng Cai[1†], Hao Liang[1†], Chang Xu[1†], Tao Xie, Wentao Zhang[2*], Bin Cui[2*]
Peking University, Beijing, China
{wentao.zhang, bin.cui}@pku.edu.cn

***Abstract*—The data-centric paradigm has emerged as a pivotal direction in artificial intelligence (AI), relying on high-quality training data. This shift is especially critical in the Text-to-SQL task, where model performance is constrained by the scarcity, limited diversity, and structural simplicity of existing datasets. To address these challenges, we propose TEXT2SQL-FLOW, a SQL-aware data augmentation framework that systematically generates large-scale, semantically valid, and structurally diverse Text-to-SQL pairs from limited seed data. Our framework operates along six augmentation dimensions and integrates an end-to-end pipeline featuring SQL execution verification, natural language (NL) question generation, chain-of-thought (CoT) reasoning trace generation, and data classification. A modular `Database Manager` further ensures cross-database compatibility and scalability. Leveraging this framework, we construct SQLFLOW, a high-quality dataset comprising 89,544 annotated examples. We demonstrate the utility of SQLFLOW in both fine-tuning and prompt-based settings: (1) For open-source large language models (LLMs), fine-tuning with SQLFLOW enhances the problem-solving capabilities. Under the same data budget, models trained on SQLFLOW achieve competitive performance gains across multiple benchmarks. (2) For closed-source LLMs, we propose a masked alignment retrieval method that leverages SQLFLOW as both a knowledge base and the training data for the retrieval model. This approach enables structure-aware example matching by modeling fine-grained alignments between NL questions and SQL queries. Experimental results show that our retrieval strategy outperforms existing example retrieval methods, highlighting the dual importance of SQLFLOW's high-quality data and our novel retrieval technique. Our work establishes a scalable, data-centric foundation for advancing Text-to-SQL systems and underscores the indispensable role of structured, high-fidelity data in modern AI development.**

***Index Terms*—Text-to-SQL, Large Language Model, SQL**

I. INTRODUCTION

In recent years, the data-centric artificial intelligence (AI) paradigm has garnered increasing attention [1], [2]. Traditional algorithm-centric approaches primarily focus on expanding model architectures and optimizing learning algorithms. However, in many cutting-edge fields, the main bottleneck of development has gradually shifted from algorithmic complexity to the availability of high-quality data. Continuous optimization of algorithms is facing diminishing marginal returns, while vast amounts of data remain underutilized, containing immense potential value. Therefore, the data-centric perspective emphasizes the core role of high-quality, diverse, and well-structured training data in advancing artificial intelligence [3].

Taking large language models (LLMs) as an example, their generalization ability and robustness highly depend on the breadth and quality of the training data. Similarly, in downstream tasks such as domain adaptation, high-quality data can serve both as reference material for generating answers and as guidance for solving problems [4]. Consequently, building scalable and high-quality datasets has become a cornerstone for advancing artificial intelligence.

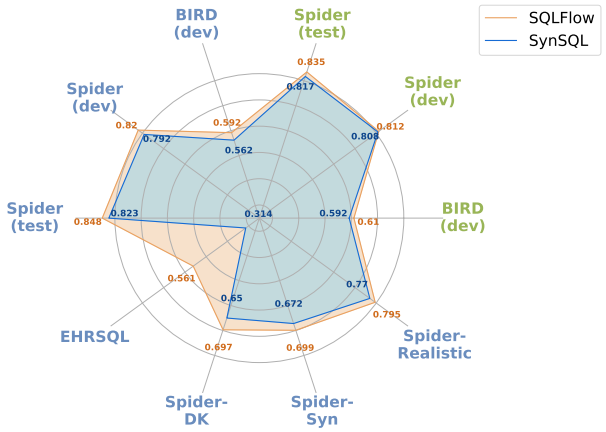

Fig. 1: Performance comparison of models trained on SQLFlow vs. SynSQL [5] (both at 89,544 samples). Blue labels indicate results for fine-tuned open-source models, while green labels indicate results of closed-source model with few-shot retrieval for prompt construction.

The Text-to-SQL task is currently facing an urgent demand for high-quality data. Although the structured query language (SQL) enables powerful and precise database interactions, its complex syntax and steep learning curve limit its accessibility for non-expert users. The Text-to-SQL task aims to automatically translate natural language (NL) questions into executable SQL queries, allowing users without specialized knowledge to access databases conveniently [6]. This capability shows broad application prospects in intelligent analysis, business intelligence platforms, and enterprise data services. With the rise of LLMs, significant progress has been made in Text-to-SQL research. Existing methods are mainly divided into two categories: *(1) fine-tuning-based methods* [7]–[10] and *(2) prompt-based methods* [11]–[14]. Although these methods have competitive performance on mainstream benchmarks, their further development is still constrained by the scarcity

[†]Equal contribution.
[*]Corresponding authors.

TABLE I: Comparison of SQL query complexity between the original seed datasets and the datasets augmented by TEXT2SQL-FLOW. Rows prefixed with "SQLFlow-" show augmented dataset stats; the bottom row shows overall SQLFLOW statistics.

Dataset	Size	Query Feature Presence per SQL %				Feature Count per SQL		
		Window Func.	Set Op.	Subquery	Aggregation	# CASE	# WHERE	# Join
Spider-train	8,659	0.00	6.07	15.59	22.6	0.00	0.86	0.71
SQLFlow-Spider	37,517	5.27	7.37	25.55	40.75	0.19	1.50	1.04
BIRD-train	9,428	0.04	0.23	6.71	9.16	0.09	1.25	0.90
SQLFlow-BIRD	37,536	4.00	4.83	20.88	29.55	0.22	1.82	1.15
EHRSQL-train	18,472	11.99	0.23	67.72	20.72	0.02	4.13	0.42
SQLFlow-EHRSQL	14,491	9.22	4.11	48.13	33.43	0.17	3.56	1.03
SQLFLOW	89,544	5.37	5.77	27.19	34.80	0.20	1.97	1.09

of high-quality data.

In both categories of methods, high-quality data plays a pivotal role within the data-centric AI paradigm. Fine-tuning-based methods rely on data pairs to enable LLMs to learn the mapping from NL questions to SQL queries, thereby enhancing the model's reasoning capability. In prompt-based methods, where model weights are inaccessible, in-context learning becomes key to improving performance, and the quality of retrieved few-shot examples is critical. Data can serve as reference examples in a knowledge base or be used to train retrieval models for obtaining more representative few-shot samples [7], [15], [16].

Since the Text-to-SQL task primarily relies on data pairs composed of NL questions and corresponding SQL queries, efficiently obtaining these data pairs becomes a critical challenge. Manual collection not only struggles to guarantee quality but is also limited in scale. Existing public datasets (e.g., Spider [17], BIRD [18]) mainly rely on manual annotation, suffering from limited size, poor scalability, and narrow data coverage. Recent studies (e.g., OmniSQL [5]) have attempted to explore synthetic data generation pathways. However, such methods are often costly, generate SQL with excessive variation, and require large data volumes to achieve performance improvements. Thus, achieving scalable, high-quality data efficiently has become a central challenge.

To address the aforementioned issues, we propose TEXT2SQL-FLOW, a robust Text-to-SQL data augmentation framework. This framework can efficiently augment original Text-to-SQL data, enhance data diversity, and possesses good scalability. Unlike synthesizing data from scratch, our framework builds upon existing SQL queries, defining six data augmentation dimensions covering *Data Value Transformations*, *Query Structure Modifications*, and *Complexity Enhancements*, among others, utilizing LLMs to synthesize diverse and semantically valid augmented SQL data. A data synthesis pipeline is designed to automate the generation of high-quality data. Various operators are responsible for SQL execution filtering, corresponding NL question generation, prompt generation, chain-of-thought (CoT) path generation for problem-solving, and data classification operators that categorize the generated augmented data to construct high-quality datasets. This framework can automatically generate large-scale SQL variants with diverse stylistic patterns, building high-quality, large-scale datasets from originally small-scale data. Furthermore, to improve the framework's adaptability to different databases and enhance database operation efficiency, a `Database Manager` module is designed. It interacts with various databases through a `Database Connector` interface. Each database inherits from and implements the methods defined in the `Database Connector`, while the `Database Manager` performs high-level invocations, thereby enhancing extensibility across different databases.

Based on our framework, a high-quality Text-to-SQL dataset comprising 89,544 entries called SQLFLOW is constructed. Using SQLFLOW, we explore the applications of enhanced data in Text-to-SQL tasks. For open-source LLMs, the augmentation framework can provide higher-quality training data. The included CoT can further assist models in improving their reasoning capabilities and problem-solving proficiency. After fine-tuning open-source models with data generated by our framework, we achieved excellent performance on multiple benchmarks, validating the framework's effectiveness. As shown in Figure 1, under the same data scale, the model trained by SQLFLOW achieves higher performance than SynSQL [5] across all benchmarks, attaining 59.2% on the BIRD-dev dataset and 83.5% on the Spider-test dataset. For closed-source models, we adopt prompt-based methods, retrieving few-shot examples to provide references for the LLM in solving problems, thereby achieving broad applicability across various scenarios. Existing systems typically select examples from limited public datasets, which have significant limitations in semantic diversity and structural coverage. Therefore, the example knowledge base for retrieval can also be constructed using the data generated by our framework. However, at the retrieval method level, current mainstream retrieval strategies often focus merely on surface-level semantic similarity, neglecting the deep syntactic and logical relationships between NL questions and SQL queries. To address this, we propose an improved masked alignment retrieval method for example selection. This method learns the masked alignment relationships between NL questions and SQL queries during

the training phase and achieves fine-grained structure-aware matching during the inference phase, enhancing the relevance and effectiveness of retrieved examples. The training data for this retrieval model also utilizes the data constructed by our framework. As shown in Figure 1 shows that our retrieval method achieves competitive performance on benchmarks, with 61.0% on the BIRD-dev set and 83.5% on the Spider-dev set, outperforming other methods.

Our contributions are summarized as follows:

- We propose TEXT2SQL-FLOW, a robust Text-to-SQL data augmentation framework that uses existing Text-to-SQL pairs as seeds to systematically generate semantically valid and structurally diverse augmented samples along six dimensions efficiently. An end-to-end automated pipeline is designed, integrating modules for SQL execution filter, NL question generation, prompt generation, CoT generation, and data classification, ensuring high-quality generated data. Furthermore, by introducing a `Database Manager` module, the framework supports flexible adaptation to various database systems, improving scalability.
- Based on TEXT2SQL-FLOW, we generate a high-quality Text-to-SQL dataset named SQLFLOW, comprising 89,544 samples. The data is efficiently generated, exhibiting high diversity and quality. Experiments on open-source LLMs show that fine-tuning with SQLFLOW enhances model performance, achieving competitive results efficiently on benchmark evaluations. Notably, it attained 59.2% on the BIRD-dev dataset and 83.5% on the Spider-test dataset, outperforming other models trained with datasets of similar scale.
- For scenarios involving closed-source models, we utilize SQLFLOW as a few-shot example knowledge base and further propose a masked alignment retrieval method. By learning fine-grained alignment relationships between NL questions and SQL queries, our method enables more accurate example matching. Experiments show that our retrieval model, trained using SQLFLOW, enhances the Text-to-SQL accuracy of closed-source models, outperforming existing retrieval methods based on semantic similarity, achieving 61.0% on the BIRD-dev dataset and 83.5% on the Spider-dev dataset.

II. RELATED WORKS

A. Text-to-SQL Techniques

Early rule-based approaches rely on handcrafted templates to map NL questions to SQL queries [19]–[21]. Although effective in domain-specific settings, these methods exhibit limited generalizability due to their dependence on rigid rule sets and poor adaptability across diverse database schemas.

The advent of neural sequence-to-sequence architectures marks a significant shift in the field [22]. Methods such as those in [23]–[26] employ encoder–decoder frameworks augmented with attention mechanisms and schema-aware graph representations to better capture structural relationships between NL and database elements. While these innovations have led to substantial performance improvements, they demonstrate limited robustness when confronted with unseen or heterogeneous schema structures.

The rise of pre-trained language models, such as BERT [27], [28] and T5 [29], further advanced Text-to-SQL research. By combining large-scale unsupervised pretraining with task-specific fine-tuning, these models effectively learn semantic alignments between NL utterances and database schemas [30], [31]. Despite their strong empirical performance, they remain heavily reliant on high-quality, human-annotated training data and are prone to overfitting, especially in low-resource or cross-domain scenarios.

Recently, the emergence of LLMs has opened new directions for Text-to-SQL. Through sophisticated prompting strategies, LLMs can generate accurate SQL queries even in zero-shot or few-shot settings [15], [32]. Techniques including prompt engineering [7], task decomposition [14], and self-correction have further enhanced their reliability and generalization capabilities, demonstrating considerable promise for cross-domain and real-world applications.

B. Data-Centric AI for Text-to-SQL

Data-Centric AI for Text-to-SQL can be divided into two main aspects. First is how to better leverage existing data. For example, schema linking structures database schema information to make prompts clearer and reduce ambiguity [13], [33]–[35], aiming at improving the model's understanding of database structures. Additionally, data preprocessing and cleaning [8], [36]–[38] are crucial, such as constructing inter-table relationship graphs, standardizing field names, and eliminating redundant or conflicting schema information, thereby providing the model with cleaner and more consistent inputs.

The second aspect concerns how to generate synthetic data more effectively. Current data synthesis approaches fall primarily into two categories. The first is SQL-to-question [39]–[42], which uses templates or syntactic rules to generate NL questions from SQL queries. This approach benefits from the precision of SQL and the fluency of language models, enabling the creation of high-fidelity $\langle$question, SQL$\rangle$ pairs. However, it struggles with complex queries and is difficult to scale due to its reliance on manually designed rules. The second category is question-to-SQL [11], [43]–[45], which first generates NL questions, either via crowdsourcing or LLMs, and then uses a Text-to-SQL model to infer the corresponding SQL. Although the generated questions tend to be more natural, this method inherits errors from the Text-to-SQL model and is prone to introducing semantic or syntactic inaccuracies. Recent research attempts to overcome these limitations. Sense [46] employs a prompt chaining strategy to guide GPT-4 through sequential generation of schema, question, and SQL, ensuring semantic consistency among the three components. However, its reliance on closed-source, high-cost models like GPT-4 limits its scalability and reproducibility. In contrast, OmniSQL [5] decouples the generation pipeline, allowing different stages to be handled by open-source or lightweight LLMs. It also

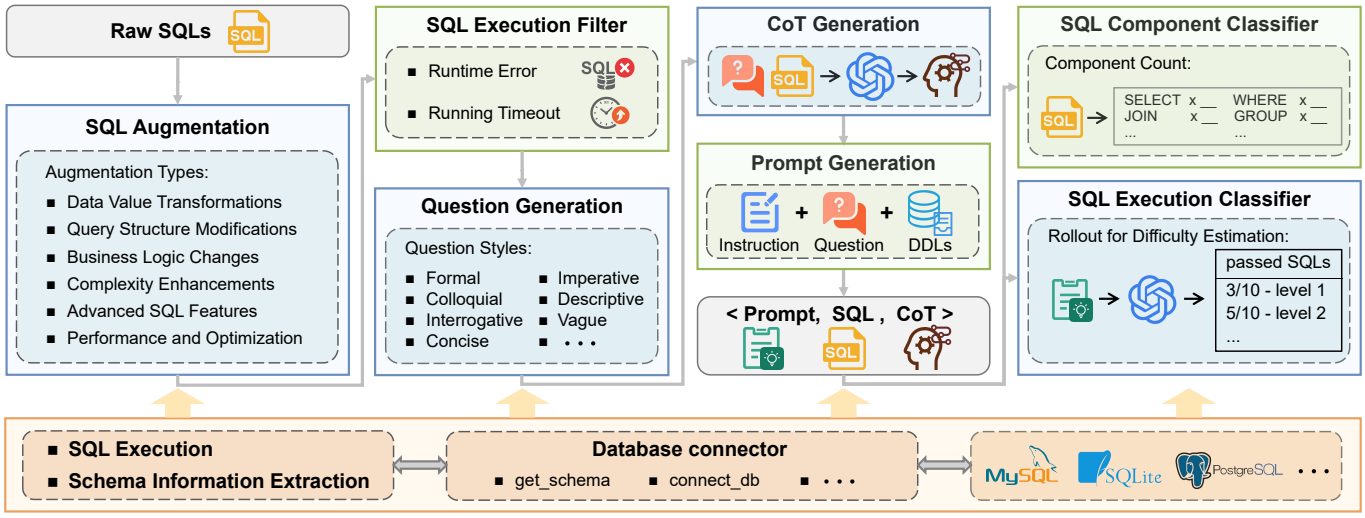

Fig. 2: Overall framework of our work.

incorporates a CoT mechanism to further enhance the logical consistency and interpretability of synthetic samples, offering a more accessible and scalable solution.

III. Methodology

In our work, we propose a data augmentation framework called Text2SQL-Flow, which consists of a data augmentation pipeline and an adapted `Database Manager` module. Part III-A introduces the data augmentation pipeline, which generates augmented SQL samples from raw seed SQL queries and further synthesizes multidimensional data pairs, as shown in the upper part of Figure 2. Part III-B presents the `Database Manager` module, as shown in the lower part of Figure 2. Through this framework, we constructed a Text-to-SQL dataset called SQLFlow. The following two parts demonstrate the utilization of SQLFlow. In Part III-C, we use CoT rationales as training targets to supervisedly fine-tune open-source LLMs, thereby enhancing their problem-solving capabilities, as shown in the left part of Figure 3. Part III-D addresses closed-source models and adopts a prompt-based approach. Specifically, we train an embedding model on SQLFlow that aligns the representations of masked NL and SQL queries, thereby improving the effectiveness of the retrieval system, as illustrated in the right part of Figure 3.

A. Data Augmentation Pipeline

*1) **SQL Augmentation**:* Data serves as the cornerstone of Data-Centric AI and plays a pivotal role in training LLMs. However, most accessible data originates from open-source datasets that are limited in scale and stylistically homogeneous. Although proprietary datasets exist, their volume rarely suffices for robust model training. Consequently, data scarcity and uniformity constrain the generalization capacity of LLMs, especially for complex SQL generation tasks.

Existing approaches to SQL data synthesis often focus on generating queries from scratch, such as via templates or random construction, overlooking the untapped potential of high-quality SQL queries already available. This limits the effective utilization of existing resources. In contrast, such high-quality queries can be repurposed through data augmentation to generate diverse yet semantically equivalent or closely related augmented SQL data. Moreover, SQL queries are inherently tied to their underlying database schemas. Constraints such as table or column names and foreign key relationships are implicitly encoded in valid SQL queries. Augmenting from existing queries naturally preserves schema consistency, ensuring that the generated augmented data remains syntactically valid and semantically aligned with the original data distribution. This yields augmented data that better reflects real-world query patterns while maintaining fidelity to the database schema.

A key challenge in data augmentation lies in producing meaningful and diverse augmented data from existing SQL queries. Generating schema-consistent and semantically coherent queries not only enriches the training data but also enhances the model's robustness and adaptability to a wide range of query formulations. To this end, we propose a data augmentation framework that leverages LLMs to generate high-quality augmented SQL data. Thanks to extensive exposure to SQL snippets during the pre-training process, LLMs have acquired strong priors over SQL syntax, semantics, and common query patterns, making them well-suited for generating syntactically valid and semantically faithful SQL data. However, a critical limitation remains: existing public SQL datasets often exhibit limited query patterns and relatively low structural complexity. When used as prompts or seeds, such simplistic queries can constrain the diversity and sophistication of the generated variants, hindering coverage of real-world usage scenarios. To overcome this, we design a set of targeted augmentation strategies that preserve the core semantics of the original queries while systematically increasing their structural complexity and stylistic variation. This enables the construction of a more comprehensive and representative Text-to-SQL dataset. Six

augmentation strategies are proposed to generate SQL queries in different directions, which can be summarized as follows:

- **Data Value Transformations**: Adjust filtering conditions, date ranges, or numeric thresholds to refine query precision; modify sorting criteria or limit values; alter aggregation boundaries (e.g., `GROUP BY` with different temporal granularities).
- **Query Structure Modifications**: Rewrite aggregation queries into window functions and vice versa; restructure simple queries into subqueries or common table expressions (CTEs); substitute `JOIN` with `EXISTS` or `IN`; switch between correlated and uncorrelated subqueries to enhance flexibility and maintainability.
- **Business Logic Changes**: Adapt queries to alternative business domains, such as shifting from sales analysis to inventory management or from customer analytics to supplier evaluation; adjust data granularity (e.g., daily data to monthly summaries); change the analysis perspective (e.g., profit analysis to cost analysis); or modify metrics (e.g., sum to average, count to percentage).
- **Complexity Enhancements**: Introduce additional filtering conditions, join extra tables, incorporate `CASE` expressions for conditional logic, or embed data validation and quality checks to increase query complexity.
- **Advanced SQL Features**: Utilize partitioned window functions, `UNION/INTERSECT/EXCEPT` operations, recursive CTEs, and pivot/unpivot operations to broaden the technical scope of queries.
- **Performance and Optimization**: Improve execution efficiency by adding optimization hints, restructuring queries to leverage indexes, adopting efficient query patterns, and optimizing `WHERE` clauses.

To ensure the validity of the generated augmented SQL queries, we incorporate randomly sampled database values into the prompts to improve semantic grounding. Each prompt consists of the following components: *(1) Instruction* (I_{sql}): a directive guiding the LLM to generate SQL queries that satisfy realistic analytical requirements; *(2) Database Schema* (S): the full set of `CREATE TABLE` statements describing all relations in the database; *(3) Database Values* (V): a set of randomly sampled (column, value) pairs drawn from the underlying tables to instantiate meaningful query predicates; *(4) Original SQL query* (s_i^{orig}): the query to be augmented; and *(5) Augmentation Direction* (c): one of six predefined augmentation strategies. Let $C = \{c_1, \ldots, c_6\}$ denote the set of augmentation strategies. For each original query s_i^{orig}, we uniformly sample a strategy $c \sim \text{Uniform}(C)$, draw representative database values V from the relevant tables, and construct an augmentation prompt $P_{\text{aug}}(I, S, s_i^{\text{orig}}, V, c)$. The augmented query is then produced by the language model as:

$$s_i^{\text{aug}} = \text{LLM}\big(P_{\text{aug}}(I_{\text{sql}}, S, V, c, s_i^{\text{orig}})\big) \tag{1}$$

where $P_{\text{aug}}(\cdot)$ denotes the deterministic prompt construction process, and $\text{LLM}(\cdot)$ represents the stochastic query generation by the language model.

*2) **SQL Execution Filter**:* Not all generated SQL queries are valid or efficient. Therefore, it is necessary to filter the SQL queries. An SQL execution filter is applied to screen out unsuitable candidates. The filter evaluates queries along two dimensions: *(1) Executability*, which verifies that the SQL syntax is correct and the query can be successfully executed on the target database; and *(2) Execution Efficiency*, which discards queries whose runtime exceeds a predefined threshold to ensure system responsiveness. After applying this filter, all resulting SQL queries are guaranteed to be valid.

*3) **Question Generation**:* After obtaining the valid augmented SQL queries, the next step is to generate semantically equivalent NL questions. To ensure linguistic diversity in the synthetic data, it is essential to incorporate a wide range of stylistic instructions. Inspired by OmniSQL [5], we analyze real-world user queries along multiple linguistic dimensions and identify eleven common stylistic categories, grouped into four high-level aspects:

- **Tone and Formality**: formal and colloquial
- **Syntactic Structure and Intent**: imperative, interrogative, and declarative
- **Information Density and Clarity**: concise, descriptive, vague, and metaphorical
- **Interaction Patterns**: role-playing and procedural

The first two categories capture cases where user intent is unambiguous but expressed through varying tones or syntactic forms. In contrast, vague and metaphorical styles involve ambiguous or figurative language, often requiring external or contextual knowledge for accurate interpretation.

To synthesize NL questions, we design structured prompts for LLMs comprising the following four components: *(1) Instruction* (I_{nl}): a directive instructing the LLM to translate a given SQL query into an NL question; *(2) Augmented SQL Query* (s_i^{aug}): the SQL query to be translated; *(3) Database Schema* (S): the complete schema provided as `CREATE TABLE` statements; and *(4) Target Language Style* (τ): a stylistic variant randomly sampled from a predefined set $\mathcal{T} = \{\tau_1, \ldots, \tau_{11}\}$, where each style is defined with a description and illustrative examples.

For each SQL query s_i, we sample a style $\tau \sim \text{Uniform}(\mathcal{T})$ and assemble the prompt as $P_{\text{nl}}(I, s_i^{\text{aug}}, S, \tau)$. The LLM then generates an NL question:

$$q_i = \text{LLM}\big(P_{\text{nl}}(I_{\text{nl}}, S, \tau, s_i^{\text{aug}})\big). \tag{2}$$

To ensure both semantic fidelity and linguistic diversity, we generate k candidate questions $\{q_{i1}, \ldots, q_{ik}\}$ for each s_i^{aug} by independently sampling a new style τ for each generation. Among these candidates, we compute pairwise semantic similarities and retain the question with the highest average similarity to the others, thereby preserving semantic consistency while encouraging stylistic variation. The final synthetic dataset is constructed as $\mathcal{D} = \{(q_i, s_i^{\text{aug}})\}_{i=1}^{N}$, where N is the number of distinct SQL queries. This approach yields a dataset with rich linguistic variation and broad semantic coverage, enhancing both robustness and generalization in downstream applications.

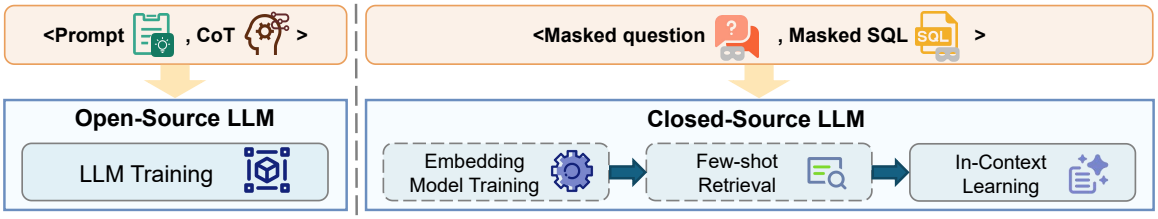

Fig. 3: Augmented data utilization for open-source and closed-source LLMs

*4) **Chain-of-Thought Generation**:* Chain-of-Thought (CoT) reasoning enhances a model's ability to solve complex tasks by decomposing them into a sequence of smaller, manageable subproblems, thereby enabling step-by-step reasoning. Equipping LLMs with CoT reasoning capabilities thus improves their overall reasoning performance. In the context of Text-to-SQL translation, the CoT reasoning process typically involves the following steps: (1) analyzing the intent of the NL question, (2) interpreting the database schema, (3) identifying relevant tables and columns, (4) filtering necessary information, (5) selecting appropriate SQL operators, and (6) incrementally constructing the final SQL.

To generate CoT reasoning traces, we employ an LLM with strong reasoning capabilities. The prompt design typically includes *(1) instruction*(I_{cot}), *(2) database schema*(S), *(3) generated NL question*(q_i), and *(4) augmented SQL query*(s_i^{aug}). Guided by this prompt for CoT generation, the LLM produces a complete reasoning chain that encompasses intermediate reasoning steps as well as the final SQL query, which can be expressed as follows:

$$cot_i = \text{LLM}\big(P_{\text{cot}}(I_{\text{cot}}, S, s_i^{\text{aug}}, q_i)\big). \tag{3}$$

where $P_{\text{cot}}(\cdot)$ denotes the process of CoT generation. During validation of the CoT, the generated SQL query is extracted from the reasoning chain. The CoT process is accepted as a valid solution only if the execution result of the generated SQL matches that of the reference SQL on the given database.

*5) **Prompt Generation**:* The prompt (p_i) serves as the primary input to the model, containing all necessary information for it to perform reasoning. To facilitate reliable text-to-SQL generation, a well-structured prompt incorporates not only the NL question (q_i) but also the database schema (S) and a specific task instruction (I_{prop}) that guides the model. The composition process is formally defined as:

$$p_i = P_{\text{prop}}(I_{\text{prop}}, S, q_i) \tag{4}$$

where $P_{\text{prop}}(\cdot)$ denotes the function that synthesizes the final prompt from its constituent parts.

*6) **SQL Component Classifier**:* Classifying generated SQL queries facilitates a deeper analysis and understanding of their structural complexity. Following the evaluation criteria established in the Spider benchmark, we categorize SQL queries into four difficulty levels: *easy*, *medium*, *hard*, and *extra hard*. This classification is primarily based on the number and complexity of syntactic components present in the query, including column selections and the use of aggregate functions in the `SELECT` clause, as well as the presence of keywords such as `GROUP BY`, `ORDER BY`, `INTERSECT`, or advanced constructs like nested subqueries. Generally, queries incorporating a greater number of such components are considered structurally more complex and thus assigned a higher difficulty level. Under this principle, the augmented SQL query s_i^{aug} can be assigned a reasonable and quantifiable component-based difficulty. This process is formally expressed as:

$$cd_i = \text{CC}(s_i^{\text{aug}}) \tag{5}$$

where $\text{CC}(\cdot)$ denotes the mapping function of the Component Classifier, and cd_i represents the component difficulty of the query.

*7) **SQL Execution Classifier**:* It is important to note that structural complexity alone does not fully determine the overall difficulty of an SQL query: some queries with numerous components may be logically straightforward and thus easier for models to generate correctly. From the perspective of LLMs, a more meaningful measure of difficulty is whether the model can consistently produce the correct SQL for a given NL question in the Text-to-SQL task. To this end, we introduce the *execution difficulty* metric ed_i. Specifically, we prompt the LLM to perform the Text-to-SQL task k times for the same input prompt p_i and count the number of successful executions n. The execution difficulty is then defined as:

$$ed_i = 1 - \frac{n}{k} \tag{6}$$

This metric reflects the practical capability of the current LLM in solving a specific Text-to-SQL problem. Unlike the component-based classifier, execution difficulty is model-dependent. More capable LLMs will achieve higher success rates on the same task, thereby exhibiting lower execution difficulty. Consequently, this metric provides a dynamic and performance-oriented perspective on task difficulty, better aligned with real-world generation behavior.

*8) **Augmented Data Overview**:* Leveraging the proposed Text-to-SQL data augmentation pipeline, TEXT2SQL-FLOW, we generate an augmented dataset from the original SQL-query pairs s_i^{ori}. The pipeline systematically synthesizes diverse and valid Text-to-SQL examples, which can serve as a valuable supplement to existing Text-to-SQL datasets. The final dataset comprises N entries of the following form:

$$\{s_i^{\text{aug}},\ q_i,\ S,\ p_i,\ \text{cot}_i,\ \text{ed}_i,\ \text{cd}_i\}_{i=1}^{N}$$

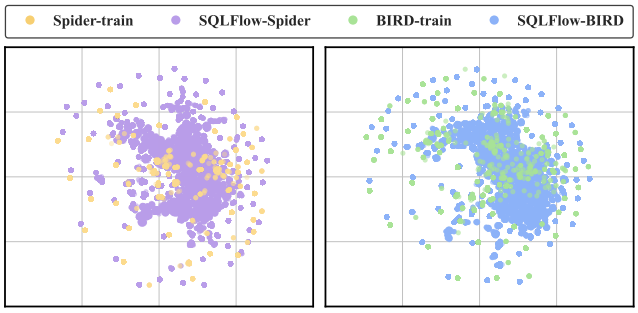

Fig. 4: Comparison of visualization between original SQLs and augmented SQLs using our framework.

Using TEXT2SQL-FLOW, we construct a dataset named SQLFLOW. The raw seed data is sourced from widely used benchmarks. The final dataset comprises a total of 89,544 entries: 37,517 derived from the Spider-train dataset [17], 37,536 from the BIRD-train dataset [18], and 14,491 from the EHRSQL-train dataset [47]. In this paper, we refer to these subsets as SQLFlow-Spider, SQLFlow-BIRD, and SQLFlow-EHRSQL, respectively. It should be noted that these data are augmented and do not overlap with the original training datasets. Our approach augments existing public datasets, thereby enriching their diversity and coverage. It is scalable, efficient, and capable of generating high-quality synthetic data. As shown in Table I, our pipeline not only mitigates the sparsity and structural gaps inherent in the original dataset but also substantially extends its complexity frontier—surpassing the original in both scale and quality. These improvements are visually illustrated in Figure 4, which presents a 2D t-SNE scatter plot generated from feature vectors derived from the data summarized in Table I.

B. Database Manager System Design

Part III-A describes the data augmentation pipeline; however, within the framework, an efficient and reliable database interaction mechanism serves as the core infrastructure underpinning the stable operation of the entire pipeline. Specifically, multiple critical steps within the pipeline, such as SQL execution filtering and SQL augmentation, require access to the underlying database system. These interactions can be categorized into two types: (1) *SQL Execution* and (2) *Schema Metadata Extraction*.

SQL Execution involves submitting generated SQL queries to the database engine and retrieving the execution feedback, which is primarily used to validate the actual executability of the SQL queries. However, mainstream database systems, such as SQLite, MySQL, and PostgreSQL, exhibit significant differences in their driver interfaces and connection protocols. Directly invoking each database's native API not only complicates cross-platform compatibility but also necessitates maintaining separate adaptation logic for every system, leading to code redundancy and severely limiting the system's scalability.

Schema Metadata Extraction aims to retrieve structured metadata about the database, including table definitions like `CREATE TABLE` statements, column attributes (data type, nullability, default values), primary/foreign key constraints, index information, and representative `INSERT` statements that can be used for SQL augmentation. Such metadata is foundational for understanding database semantics, generating syntactically and semantically valid SQL queries, and implementing context-sensitive data augmentation strategies. Although most relational databases provide mechanisms, such as querying `INFORMATION_SCHEMA` or system catalog tables, to retrieve schema information, the query syntax, field naming conventions, and data formats vary significantly across systems, further complicating unified handling. Moreover, for a given database instance, the schema typically remains static over short time intervals; repeatedly issuing identical metadata queries introduces unnecessary I/O overhead and latency, substantially degrading the overall throughput of the pipeline.

To systematically address these challenges, we designed and implemented a `Database Manager` module within the underlying architecture of the data augmentation framework. This module abstracts away low-level database interaction details and provides upper-layer components with a unified, efficient, and extensible programming interface. Specifically, the Database Manager supports `batch_sql_execution` and `batch_compare_sql`, improving processing throughput under high-concurrency scenarios. Additionally, it encapsulates schema metadata retrieval logic and exposes high-level methods such as `get_ddl` (to obtain Data Definition Language) and `get_insert_statement` (to generate representative insertion statements), thereby simplifying upper-layer dependencies on database structure.

To achieve true cross-database compatibility, we introduced an abstract base class `DatabaseConnector`. This class explicitly defines a standardized interface, including methods such as `connect_db` (establish connection and perform initialization), `execute_sql` (execute arbitrary SQL and return results), and `get_schema` (extract complete schema metadata). For each specific database system, one only needs to inherit from this base class and implement its database-specific driver calls and error-handling logic (e.g., `SQLiteDatabaseConnector`). This enables seamless integration into the system. At runtime, the Database Manager dynamically instantiates the appropriate connector based on configuration, while its core logic remains agnostic to the underlying database type, thus achieving the architectural goal of "design once, adapt to many databases."

To further enhance overall throughput, the Database Manager incorporates a cache-based schema management mechanism. Upon the first request for a database's schema, the system serializes its structured metadata and stores it in an in-memory cache. Subsequent requests for the same schema are served directly from the cache, eliminating redundant queries and reducing I/O load and response latency. This mechanism is especially effective in large-scale SQL generation and validation tasks, where hundreds to thousands of repeated schema queries against the same database are common.

C. Synthetic Data for Open-Source Models

For open-source LLMs, their reasoning capabilities can be enhanced through supervised fine-tuning (SFT), a process in which the model learns to map inputs to outputs. In the case of augmented data from TEXT2SQL-FLOW, each instance includes a prompt p_i and a corresponding CoT cot_i. The pairs $\langle p_i, cot_i \rangle$ extracted from the SQLFLOW dataset serve as high-quality training data $\mathcal{D}$. During SFT, we train on the full output sequence using standard causal language modeling:

$$\mathcal{L} = -\mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\sum_{t=1}^{|y|} \log P_\theta(y_t \mid x, y_{<t}) \right] \tag{7}$$

where the input x corresponds to the prompt p_i, and the output y represents the complete CoT sequence cot_i. The model parameters are denoted by θ. This loss function, applied over the entire output sequence, encourages the model not only to generate correct SQL queries but also to internalize interpretable and structured reasoning pathways.

D. Synthetic Data for Prompting-based Methods

For closed-source LLMs, whose parameters are not accessible, prompt-based methods have become a crucial technique for improving performance. A typical prompt provided to an LLM generally includes multiple components: an instruction, database schema information, and the NL question posed by the user. To enhance the model's generalization ability across different questions, few-shot examples are often retrieved and incorporated into the prompt context in practice. These examples are typically selected due to their similarity to the target question, serving as guidance for the model in generating the corresponding SQL query. Each few-shot example consists of an NL question paired with its corresponding SQL query. Ideally, such examples should effectively reveal the structural patterns and generation logic of the target SQL, thereby improving the model's reasoning performance.

Two problems need to be considered during few-shot example retrieval: *(1) What information is available during retrieval?* During retrieval, only the user's NL question is available, and we cannot know the ground-truth SQL query in advance. Moreover, any intermediate representation derived from this NL question may introduce information loss, thereby degrading retrieval quality. *(2) Which kind of examples do we need?* The primary value of few-shot examples lies in providing structural guidance for SQL generation. Therefore, the SQL queries of the retrieved examples should exhibit high similarity to the target query, but not necessarily the higher the similarity, the better. Notably, different components of the NL question and SQL query contribute unequally to the similarity. Specifically, the "skeleton" of a question reflects its basic querying pattern, while the skeleton of an SQL query manifests as the logical composition of core clauses such as "SELECT–FROM–WHERE–JOIN". These skeletal elements encode the query's logical intent and functional structure, serving as critical signals for guiding model generation. In contrast, fine-grained details, such as specific table names, column names, string literals, or numeric constants, are highly dependent on the particular database schema and application context. If directly used in similarity computation, they often introduce noise and impair retrieval effectiveness. Thus, an ideal retrieval mechanism should identify examples whose corresponding SQL queries are structurally aligned with the target query, based solely on the NL question.

However, existing methods for retrieving few-shot examples fail to provide effective guidance because similarity has not been sufficiently exploited. In this case, we propose a structure-aware few-shot retrieval method for Text-to-SQL. The core idea is to map both NL questions and SQL queries into a unified, detail-abstracted semantic space, where retrieval is performed based on structural similarity. Specifically, inspired by the masking strategy in DAIL-SQL [7], we apply standardized masking to schema-dependent elements (e.g., table/column names), string literals, and numeric constants in both questions and SQL queries, denoted as $\text{Mask}(q)$ and Mask(s), respectively. Building upon this, we develop a structure-aware embedding model based on an LLM to compute semantic similarity between masked NL questions and masked SQL queries. Given a target question q_t, we retrieve the top-k examples from a knowledge base $B = \{(q_i, s_i)\}_{i=1}^{N}$ whose masked SQL representations $\text{Mask}(s_i)$ are closest to $\text{Mask}(q_t)$ in the embedding space, and use the original (q_i, s_i) pairs as the final few-shot prompts. The knowledge base B is generated based on the SQLFLOW dataset we construct.

We fine-tune our embedding model using SFT and employ a contrastive learning framework for training. The training data consist of numerous (q_i, s_i) pairs. For each sample, the NL question q_i serves as the query, its corresponding SQL s_i as the positive example, and k randomly sampled SQL queries from the knowledge base (excluding s_i) as negative examples. The model is trained by optimizing the InfoNCE loss:

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(q, s^+)/\tau)}{\exp(\text{sim}(q, s^+)/\tau) + \sum_j \exp(\text{sim}(q, s_j^-)/\tau)} \tag{8}$$

where $\mathbf{e}_q^i = \text{Embed}(\text{Mask}(q_i))$ and $\mathbf{e}_s^i = \text{Embed}(\text{Mask}(s_i))$ denote the embedding vectors of the masked question and its corresponding SQL, respectively; $\text{sim}(\cdot,\cdot)$ is the cosine similarity function; and τ is a temperature parameter.

In our implementation, we adopt the Qwen3-Embedding architecture [48], employing a causal-attention-based LLM as the embedding encoder. Both the knowledge base and the training data for the embedding model are derived from the SQLFLOW dataset we constructed. The input format concatenates the task instruction with the masked question as follows: $\{\texttt{Instruction}\}$ $\{\texttt{Mask}(q_i)\}$ followed by an [EOS] token. The final sentence embedding is obtained from the hidden state of the [EOS] token in the last layer of the encoder. Our method retrieves few-shot examples that are highly aligned with the target query in SQL structure. This enhances the few-shot reasoning capability of closed-source LLMs on the Text-to-SQL task.

IV. EXPERIMENTS

A. Experimental Setup

1) Benchmarks: Multiple benchmarks are used to evaluate the effectiveness of our method.

- **Spider** [17] contains 10,181 NL questions and 5,693 unique SQL queries, involving 200 databases across multiple tables covering 138 different domains. We utilize both its development set, containing 1,034 samples, and its test set with 2,147 samples for evaluation.
- **BIRD** [18] includes 12,751 NL question-SQL pairs and 95 databases across 37 domains, focusing on large-scale noisy data and external knowledge reasoning. Our experiments are performed on its development set, which consists of 1,534 data samples.
- **EHRSQL** [47] includes approximately 24,000 NL question-SQL pairs linked to 2 open-source electronic health record databases. The dataset introduces challenges such as complex, time-sensitive questions and the inclusion of unanswerable questions.
- **Spider-DK, Spider-Syn, and Spider-Realistic** [49]–[51] are three widely-adopted robustness benchmarks designed to evaluate model performance in practical Text-to-SQL applications. Spider-DK tests the model's ability to understand implicit domain knowledge in NL questions, offering 535 samples for evaluation. Spider-Syn replaces schema-related words with manually selected synonyms and provides 1,034 evaluation samples. Spider-Realistic removes or paraphrases explicit mentions of column names, offering 508 samples for evaluation.

2) Baselines: Primarily, we evaluate the model trained on our SQLFLOW dataset against a range of both open-source and closed-source mainstream LLMs. The closed-source models include GPT-4o-mini [52], GPT-4o [53], and GPT-4-Turbo [54]. The open-source models encompass DeepSeek-Coder-7B-Instruct [55], Qwen2.5-Coder-7B-Instruct [56], Qwen2.5-7B-Instruct [57], OpenCoder-8B-Instruct [58], Meta-Llama-3.1-8B-Instruct [59], Granite-8B-Code-Instruct [60], and Granite-3.1-8B-Instruct [61]. All these open-source models have parameter counts of either 7B or 8B, matching the scale of our trained model. As for the training data of the models, we utilize not only the public datasets Spider-train [17] and BIRD-train [18], but also the synthetic dataset SynSQL proposed by OmniSQL [5]. To increase the diversity of our comparisons, for both SynSQL and SQLFLOW, regardless of their original data sources, we randomly sample two subsets: one containing 50K examples, which constitutes a suitably large scale for comparison, and another containing 90K examples (precisely 89,544 examples), matching the exact size of the SQLFLOW dataset for ease of reference. Additionally, we also use the full SynSQL-2.5M dataset, comprising 2.5 million examples, as a reference.

Secondly, for the few-shot example retrieval, we compare our retrieval method with classic example retrieval strategies. All similarity calculations use cosine similarity.

- **Zero-shot** [62] refers to a setting in which no few-shot examples are used during inference.
- **Random Selection** means selecting the few-shot examples from the knowledge base randomly without any criteria.
- **Question Similarity Selection** [15] identifies relevant examples by measuring the similarity between NL questions. It calculates the similarity between the target question and all candidate questions and selects those with the highest similarity.
- **Masked Question Similarity Selection** [63] preprocesses the target question and all candidate questions by replacing domain-specific entities—such as table names, column names, and literal values—with a generic $\langle \text{mask} \rangle$ token to mitigate the influence of domain-specific information, and then computes their similarity.
- **DAIL Selection** [7] retrieves examples by considering similarity in both the question and query spaces. It begins by masking domain-specific words in the target and candidate questions. The candidates are then ranked by question similarity. This ranking is subsequently filtered, retaining only examples whose query Jaccard similarity to the target query exceeds a predefined threshold. Since the question initially lacks a corresponding SQL, a SQL is pre-generated to serve as a reference.

3) Evaluation Metric: Execution accuracy(EX) is adopted in our experiments. This metric compares the execution result of the predicted SQL query against that of the ground-truth SQL query on a live database instance.

4) Implementation Details: To validate the effectiveness of our framework, we conduct full-parameter SFT on two widely adopted open-source LLMs: Meta-Llama-3.1-8B-Instruct [59] and Qwen2.5-Coder-7B-Instruct [56]. The training configuration includes a learning rate of 2×10^{-5}, 2 training epochs, and a warmup ratio of 15%. During LLM inference, we explore two decoding strategies: greedy decoding and majority voting. Greedy decoding (denoted as `Gre` in tables) uses a temperature of 0 to produce deterministic outputs. Majority voting (denoted as `Maj` in tables) samples 8 candidate responses per input at a temperature of 0.8. From these candidates, valid SQL queries are extracted and executed; the final prediction is selected as the query whose execution result receives the most votes.

As for the retrieval module, we develop an embedding model by full-parameter SFT a Qwen3-Embedding-0.6B model [48]. The model is trained for 3 epochs with a learning rate of 6×10^{-6}, and a per-device batch size of 2. During training, we treat the NL question as the query and its corresponding SQL as the positive example. For negative examples, we randomly sample 30 SQL queries from the rest of the dataset. For inference, we compute Question-SQL similarity by generating embeddings with both the original and our fine-tuned embedding model, using cosine similarity. We apply the masking strategy of DAIL [7] during inference and training. A 5-shot setting is used during LLM inference. We use the GPT-4o as the closed-source LLM during the experiment with greedy decoding and set the temperature to zero. We follow

TABLE II: Performance of LLMs on mainstream benchmarks. The first two blocks list closed-source and open-source base models, respectively. The last two blocks show fine-tuned models, where the first column indicates the training data setting.

LLM / Training Data	Spider dev		Spider test		BIRD dev		EHRSQL		Spider-DK		Spider-Syn		Spider-Realistic		Average	
	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj
Closed-source LLMs																
GPT-4o-mini	70.4	71.0	82.4	83.7	58.8	61.5	37.9	43.1	73.3	74.4	60.5	61.6	64.4	66.7	64.0	66.0
GPT-4-Turbo	72.4	72.2	83.4	84.2	62.0	63.6	43.1	44.8	72.3	72.1	62.9	63.5	67.5	68.3	66.2	67.0
GPT-4o	70.9	70.7	83.2	84.9	61.9	64.0	44.9	45.5	72.9	73.5	59.6	62.3	66.5	66.7	65.7	66.8
Open-source LLMs																
DeepSeek-Coder-7B-Instruct	63.2	63.2	70.5	73.2	43.1	48.0	28.6	33.9	60.9	64.1	49.9	51.7	58.7	58.9	53.6	56.1
Qwen2.5-Coder-7B-Instruct	73.4	77.1	82.2	85.6	50.9	61.3	24.3	36.9	67.5	73.6	63.1	66.9	66.7	70.5	61.2	67.4
Qwen2.5-7B-Instruct	65.4	68.9	76.8	82.6	46.9	56.4	20.9	32.1	63.7	71.8	54.2	60.0	56.7	63.6	54.9	62.2
OpenCoder-8B-Instruct	59.5	59.5	68.3	70.1	37.5	45.3	21.9	29.9	62.6	64.7	46.0	46.1	49.0	49.4	49.3	52.1
Meta-Llama-3.1-8B-Instruct	61.8	67.7	72.2	78.5	42.0	53.1	24.6	33.7	62.6	69.9	53.1	59.3	57.5	61.0	53.4	60.5
Granite-8B-Code-Instruct	58.5	59.2	64.9	68.6	27.6	32.5	16.0	22.6	50.7	54.4	45.0	46.8	48.8	49.4	44.5	47.6
Granite-3.1-8B-Instruct	58.3	65.0	69.8	75.3	36.0	47.2	19.6	32.3	60.0	66.5	47.7	53.8	46.5	57.1	48.3	56.7
Trained on Meta-Llama-3.1-8B-Instruct																
SynSQL(50k)	67.1	73.9	72.7	78.6	49.1	55.2	33.6	40.8	63.8	66.1	59.6	63.5	69.3	71.6	59.3	64.2
SynSQL(90k)	68.2	74.6	73.4	78.5	51.1	54.9	31.8	38.0	61.8	67.4	58.9	63.6	69.0	70.9	59.2	64.0
SynSQL(2.5M)	70.6	73.7	78.3	82.5	58.9	62.0	35.1	37.0	72.3	74.7	61.0	63.1	67.9	69.4	63.4	66.1
Spider+BIRD+SQLFLOW	74.9	79.2	78.4	82.3	53.4	58.9	28.4	36.5	67.7	69.7	66.6	69.1	74.4	75.0	63.4	67.2
SQLFLOW(50k)	69.9	76.8	75.1	80.1	51.4	57.6	28.0	36.4	65.9	68.1	61.3	67.5	69.6	73.5	60.2	65.7
SQLFLOW	71.4	76.4	75.8	80.0	54.6	56.8	55.5	56.3	66.5	67.7	61.6	67.3	71.4	72.7	65.3	68.2
Trained on Qwen2.5-Coder-7B-Instruct																
SynSQL(50k)	77.1	82.1	81.8	84.8	54.0	59.3	33.1	44.1	67.1	69.5	68.0	70.6	77.2	80.3	65.5	70.1
SynSQL(90k)	79.2	83.1	82.3	84.4	56.2	59.4	31.4	41.4	65.0	70.7	67.2	70.7	77.0	79.9	65.5	69.9
SynSQL(2.5M)	81.2	81.6	87.9	88.3	63.9	66.1	34.9	40.0	76.1	77.8	69.7	69.6	76.2	78.0	70.0	71.6
Spider+BIRD+SQLFLOW	85.5	87.5	87.5	88.5	58.3	64.0	27.9	39.8	71.0	73.1	75.0	76.2	82.3	83.7	69.6	73.3
SQLFLOW(50k)	80.9	84.9	84.6	85.8	57.9	62.5	27.8	39.4	69.7	71.2	70.0	74.0	77.8	82.1	67.0	71.4
SQLFLOW	82.0	85.0	84.8	86.0	59.2	61.5	56.1	58.7	69.7	71.0	69.9	74.4	79.5	81.7	71.6	74.0

the DAIL organization strategy when organizing the prompts.

All experiments are conducted on an Ubuntu 22.04.4 LTS server equipped with eight NVIDIA H200 GPUs. LLM SFT is implemented using Llama-Factory v0.9.3, while retrieval model SFT is carried out with SWIFT v3.6.3. Inference for both models is handled by vLLM v0.10.0.

5) Training Data: For open-source LLMs, we use the entire SQLFLOW dataset as training data. To train the embedding model, we use a filtered subset of the SQLFlow-Spider and SQLFlow-BIRD datasets (described in Section III-A8), retaining only SQL queries that begin with a `SELECT` statement. This yields 67,570 samples, which we refer to as SQLFLOW-PART. We also establish two baseline datasets for comparison. The first combines the standard Spider-train and BIRD-train benchmarks. The second, denoted SYNSQL-PART, is a random sample of 67,570 queries from SynSQL-2.5M [5], matched in size to SQLFLOW-PART for fair comparison. Table IV summarizes the used training datasets.

B. Performance of LLM Fine-tuning

As shown in Table II, the data generated by our framework leads to consistent performance improvements across multiple mainstream benchmarks, demonstrating the effectiveness of TEXT2SQL-FLOW. We fine-tune two widely used open-source models, Meta-Llama-3.1-8B-Instruct and Qwen2.5-Coder-7B-Instruct, to evaluate the general applicability of our data. In both cases, models trained on our generated data significantly outperformed their respective baselines as well as other models. However, due to inherent differences in model capabilities, Meta-Llama-3.1-8B-Instruct performed slightly worse than Qwen2.5-Coder-7B-Instruct. When fine-tuned with SQLFLOW data, Qwen2.5-Coder-7B-Instruct achieved notable gains: execution accuracy ("Gre") on Spider-dev increased from 73.4% to 82.0% (+8.6%), on BIRD-dev from 50.9% to 59.2% (+8.3%), and on the challenging EHRSQL benchmark from 24.3% to 56.1% (+21.8%). These results confirm two key points: (1) SFT substantially enhances model performance on Text-to-SQL tasks, and (2) the data generated via our TEXT2SQL-FLOW framework exhibits high quality and strong training utility. The consistent improvements across two distinct model architectures demonstrate the broad effectiveness of our dataset in boosting model performance.

In comparison with other training datasets, our data also demonstrates clear advantages. On comparable data scales (50K and 90K), our approach further exhibits superior performance compared to SynSQL. Specifically, on the Spider-test and BIRD-dev datasets, the model trained with SQLFLOW(50K) achieves 84.6% and 57.9% execution ac-

TABLE III: Performance comparison of retrieval strategies. "Upper limit": retrieval using ground-truth SQL. "Mask": whether query/example masking is applied. "API": whether LLM API cost is incurred during retrieval.

Retrieval Strategy	Mask	API	BIRD (dev)	Spider (dev)	Spider (test)
Baseline reference strategies					
Zero-shot	-	-	52.5	74.1	75.5
Random selection	✘	✘	56.6	78.3	79.8
Question similarity	✘	✘	58.3	79.6	81.9
	✔	✘	59.0	81.0	83.1
DAIL selection	✘	✔	58.8	80.0	82.2
	✔	✔	59.4	80.8	82.9
Question-SQL similarity	✘	✘	58.7	80.9	81.7
Upper limit	✘	✘	**61.8**	**83.9**	**85.7**
Question-SQL similarity trained on Qwen3-embedding-0.6B					
SQLFlow-part+Spider+BIRD	✘	✘	59.3	81.0	82.1
	✔	✘	<u>61.0</u>	<u>81.2</u>	<u>83.5</u>

TABLE IV: Explanation of training data used.

Data Name	Description	Data Size
Spider+BIRD	Training sets from Spider and BIRD.	18,087
SQLFLOW	Combination of SQLFlow-Spider, SQLFlow-BIRD, and SQLFlow-EHRSQL	89,544
SQLFlow-part	Filtered subset retaining only `SELECT`-initiated SQL queries from SQLFlow-Spider and SQLFlow-BIRD	67,570
SynSQL-part	Randomly sampled subset from SynSQL-2.5M, maintaining the same data volume as SQLFlow-part.	67,570

curacy ("Gre"), outperforming SynSQL(50K), which reaches 81.8% and 54.0%, respectively. These improvements highlight the higher quality and stronger generalization capabilities of our augmented training data. Likewise, the model trained with SQLFLOW(90K) not only surpasses the baseline models but also outperforms SynSQL(90K), reinforcing the robustness of our data generation approach. Remarkably, even when trained on a much smaller dataset, the model fine-tuned with SQLFLOW(90K) achieves performance comparable to SynSQL-2.5M on several challenging benchmarks. When incorporating real-world datasets such as Spider-train and BIRD-tain into the training data, the model achieves SOTA results across multiple benchmarks. We attribute these gains to the close alignment between the benchmark test distributions and our generated data, which enables more effective adaptation and leads to substantial performance improvements.

C. Comparison of Retrieval Strategies

We conduct a performance comparison between our proposed retrieval method and several baseline strategies, with results shown in Table III. The embedding model is trained on SQLFlow-part, Spider-train, and BIRD-train, as described in Table IV. The base model used in the experiments is Qwen3-Embedding-0.6B.

As shown in the lower part of Table III, fine-tuning the retrieval model on high-quality synthetic data from SQLFlow-part using our retrieval strategy significantly boosts performance. Both the high-quality augmented data and the masked alignment retrieval method contribute to this improvement. Specifically, our Question-SQL alignment method without masking achieves 59.3%, 81.0%, and 82.1% accuracy on BIRD-dev, Spider-dev, and Spider-test, respectively, outperforming the baseline that relies on Question-SQL similarity retrieval with the base (non-finetuned, no mask) model. This indicates that fine-tuning with augmented data enables the model to select more relevant examples, thereby enhancing the in-context learning capability of downstream closed-source models for accurate SQL generation. The masking strategy is another key contributor to this improvement. Ablation studies confirm its effectiveness: for non-finetuned methods, masking consistently improves retrieval performance (e.g., DAIL selection accuracy on BIRD-dev increases from 58.8% to 59.4%). Similarly, when we fine-tune on SQLFlow-part and use masking during training, performance improves to 61.0%, 81.2%, and 83.5% on the three benchmarks, demonstrating that masking remains essential even after fine-tuning. By masking literal values in the training data, the model learns to focus on the structural correspondence between questions and SQL queries while ignoring superficial lexical differences. This enables it to retrieve examples whose SQL patterns closely match the unseen ground-truth query based solely on the input question—in a single step. In contrast, methods like DAIL require generating an intermediate SQL query during retrieval, which incurs additional API costs and risks propagating errors from imperfect intermediate generations.

D. Comparison of Training Data for Retrieval Model

As shown in Figure 5, the model trained solely on the BIRD dataset performs reasonably well on the BIRD-dev but suffers a sharp drop in performance on the Spider-dev and Spider-test. This indicates that models trained on a single dataset struggle to generalize effectively to other task settings.

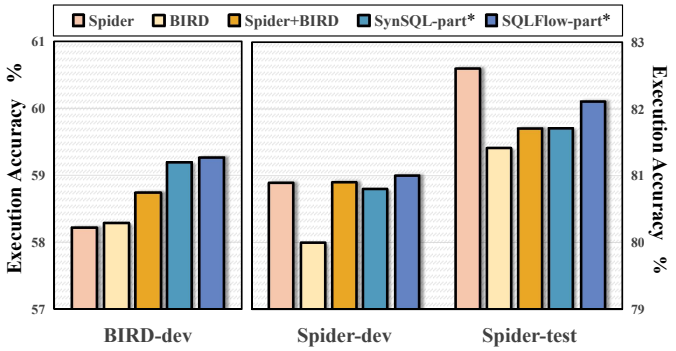

Fig. 5: Performance of question–SQL alignment strategy retrieval models trained on different datasets. No masking operations are applied during the retrieval process. "SynSQL-part*" and "SQLFlow-part*" denote the corresponding datasets combined with Spider-train and BIRD-train.

However, simply mixing data from multiple sources does not necessarily enhance generalization. For example, training on both Spider and BIRD data for the Spider benchmark even leads to degraded performance. In contrast, the model trained on the SQLFlow-part combined with Spider and BIRD achieves better results, effectively mitigating overfitting and substantially improving cross-benchmark performance. This indicates that the benefit does not stem merely from larger data volume, but from the diversity and quality of the data generated by TEXT2SQL-FLOW, which more effectively boosts model generalization. A comparison with SynSQL-part further validates this conclusion. As a fully synthetic dataset, SynSQL exhibits a data distribution that diverges significantly from real-world applications. Although it aims for generalization, its effectiveness on specific tasks remains limited, demanding massive data and computation for only modest gains. For instance, SynSQL-part achieves only slight improvement on the Spider benchmark, barely exceeding the Spider+BIRD combination. In contrast, SQLFlow-part delivers markedly better results, highlighting its superior data quality, distribution alignment, and task relevance.

E. Case Study of Few-shot Retrieval

To illustrate how our method retrieves superior examples and why these examples effectively guide the model toward generating correct SQL, we present a case study from the BIRD-dev dataset, which is shown in Figure 6. DAIL Selection's retrieval mechanism heavily relies on its pre-generated SQL. However, in this particular case, the pre-generated query is inaccurate both structurally and semantically. It fails to capture the key complex logic present in the Gold SQL, namely, returning only a single maximum-value result. Consequently, although the examples retrieved by DAIL Selection align structurally with this erroneous pre-generated query, they are entirely unrelated to the skeleton of the Gold SQL. These structurally flawed and irrelevant examples ultimately mislead the LLM, causing it to produce an incorrect answer. In contrast, our method directly aligns the question with

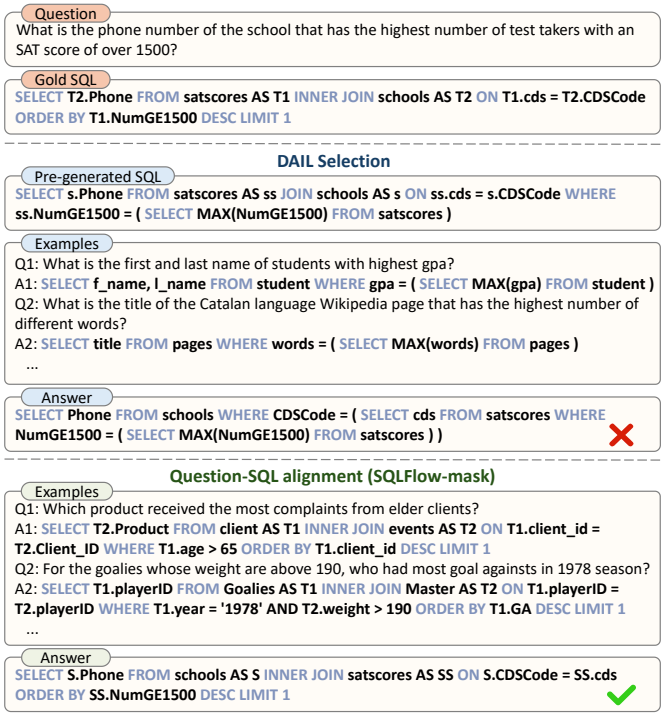

Fig. 6: Case study of few-shot retrieval.

the SQL skeleton, enabling it to retrieve examples highly similar to the Gold SQL skeleton using only the original question. As shown in the figure, the examples retrieved by our approach correctly include essential structural components such as `INNER JOIN`, `ORDER BY ... DESC`, and `LIMIT 1`. This demonstrates that our skeleton alignment mechanism captures not only syntactic similarity but also the core logical intent of the query. These high-quality examples provide the LLM with an accurate structural template, thereby guiding it to generate the correct SQL query.

V. CONCLUSION

In this work, we introduced TEXT2SQL-FLOW, a robust SQL-aware data augmentation framework that enables the scalable generation of high-quality, structurally diverse Text-to-SQL examples from limited seed data. Leveraging this framework, we constructed SQLFLOW, a large-scale dataset comprising 89,544 annotated instances. Experimental results show that models fine-tuned on SQLFLOW consistently outperform those trained on existing comparable datasets, highlighting the critical role of data quality and structural diversity in enhancing model generalization and compositional reasoning. Furthermore, our proposed masked alignment retrieval method improves the in-context learning performance of closed-source large language models by enabling structure-aware example selection. This work establishes a data-centric foundation for future Text-to-SQL research and demonstrates the importance of high-quality data in data-centric AI.

AI-Generated Content Acknowledgement

All content in this manuscript is solely authored by the human authors. The authors confirm that no generative AI tools are used in the creation, modification, analysis, or visualization of any scientific content presented in this submission.

References

[1] D. Zha, Z. P. Bhat, K.-H. Lai, F. Yang, Z. Jiang, S. Zhong, and X. Hu, "Data-centric artificial intelligence: A survey," *ACM Computing Surveys*, vol. 57, no. 5, pp. 1–42, 2025.

[2] J. Jakubik, M. Vössing, N. Kühl, J. Walk, and G. Satzger, "Data-centric artificial intelligence," *Business & Information Systems Engineering*, vol. 66, no. 4, pp. 507–515, 2024.

[3] M. Haukkala, "Data quality in artificial intelligence," 2022.

[4] K. Wang, J. Zhu, M. Ren, Z. Liu, S. Li, Z. Zhang, C. Zhang, X. Wu, Q. Zhan, Q. Liu *et al.*, "A survey on data synthesis and augmentation for large language models," *arXiv preprint arXiv:2410.12896*, 2024.

[5] H. Li, S. Wu, X. Zhang, X. Huang, J. Zhang, F. Jiang, S. Wang, T. Zhang, J. Chen, R. Shi *et al.*, "Omnisql: Synthesizing high-quality text-to-sql data at scale," *arXiv preprint arXiv:2503.02240*, 2025.

[6] A. B. Kanburoğlu and F. B. Tek, "Text-to-sql: A methodical review of challenges and models," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 32, no. 3, pp. 403–419, 2024.

[7] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *arXiv preprint arXiv:2308.15363*, 2023.

[8] Y. Gao, Y. Liu, X. Li, X. Shi, Y. Zhu, Y. Wang, S. Li, W. Li, Y. Hong, Z. Luo *et al.*, "Xiyan-sql: A multi-generator ensemble framework for text-to-sql," *arXiv preprint arXiv:2411.08599*, 2024.

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

[10] L. Sheng and S.-S. Xu, "Slm-sql: An exploration of small language models for text-to-sql," *arXiv preprint arXiv:2507.22478*, 2025.

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

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

[13] H. Li, J. Zhang, C. Li, and H. Chen, "Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 11, 2023, pp. 13 067–13 075.

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

[15] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability," *arXiv preprint arXiv:2303.13547*, 2023.

[16] L. Nan, Y. Zhao, W. Zou, N. Ri, J. Tae, E. Zhang, A. Cohan, and D. Radev, "Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies," *arXiv preprint arXiv:2305.12586*, 2023.

[17] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," *arXiv preprint arXiv:1809.08887*, 2018.

[18] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo *et al.*, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[19] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 73–84, 2014.

[20] T. Mahmud, K. A. Hasan, M. Ahmed, and T. H. C. Chak, "A rule based approach for nlp based query processing," in *2015 2nd international conference on electrical information and communication technologies (EICT)*. IEEE, 2015, pp. 78–82.

[21] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the national conference on artificial intelligence*, 1996, pp. 1050–1055.

[22] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[23] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[24] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, and D. Zhang, "Towards complex text-to-sql in cross-domain database with intermediate representation," *arXiv preprint arXiv:1905.08205*, 2019.

[25] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers," *arXiv preprint arXiv:1911.04942*, 2019.

[26] Z. Chen, L. Chen, Y. Zhao, R. Cao, Z. Xu, S. Zhu, and K. Yu, "Shadowgnn: Graph projection neural network for text-to-sql parser," *arXiv preprint arXiv:2104.04689*, 2021.

[27] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

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

[29] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[30] Y. Fu, W. Ou, Z. Yu, and Y. Lin, "Miga: a unified multi-task generation framework for conversational text-to-sql," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 11, 2023, pp. 12 790–12 798.

[31] Z. Gu, J. Fan, N. Tang, L. Cao, B. Jia, S. Madden, and X. Du, "Few-shot text-to-sql translation using structure and content prompt learning," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–28, 2023.

[32] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, J. Lin, D. Lou *et al.*, "C3: Zero-shot text-to-sql with chatgpt," *arXiv preprint arXiv:2307.07306*, 2023.

[33] M. M. H. Nahid, D. Rafiei, W. Zhang, and Y. Zhang, "Rethinking schema linking: A context-aware bidirectional retrieval approach for text-to-sql," *arXiv preprint arXiv:2510.14296*, 2025.

[34] Z. Cao, Y. Zheng, Z. Fan, X. Zhang, W. Chen, and X. Bai, "Rsl-sql: Robust schema linking in text-to-sql generation," *arXiv preprint arXiv:2411.00073*, 2024.

[35] C. Li, Y. Wang, Z. Wu, Z. Yu, F. Zhao, S. Huang, and X. Dai, "Multisql: A schema-integrated context-dependent text2sql dataset with diverse sql operations," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 13 857–13 867.

[36] M. Pourreza, S. Talaei, R. Sun, X. Wan, H. Li, A. Mirhoseini, A. Saberi, S. Arik *et al.*, "Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql," *arXiv preprint arXiv:2503.23157*, 2025.

[37] M. Glass, M. Eyceoz, D. Subramanian, G. Rossiello, L. Vu, and A. Gliozzo, "Extractive schema linking for text-to-sql," *arXiv preprint arXiv:2501.17174*, 2025.

[38] M. Kothyari, D. Dhingra, S. Sarawagi, and S. Chakrabarti, "Crush4sql: Collective retrieval using schema hallucination for text2sql," *arXiv preprint arXiv:2311.01173*, 2023.

[39] D. Guo, Y. Sun, D. Tang, N. Duan, J. Yin, H. Chi, J. Cao, P. Chen, and M. Zhou, "Question generation from sql queries improves neural semantic parsing," *arXiv preprint arXiv:1808.06304*, 2018.

[40] B. Wang, W. Yin, X. V. Lin, and C. Xiong, "Learning to synthesize data for semantic parsing," *arXiv preprint arXiv:2104.05827*, 2021.

[41] K. Wu, L. Wang, Z. Li, A. Zhang, X. Xiao, H. Wu, M. Zhang, and H. Wang, "Data augmentation with hierarchical sql-to-question generation for cross-domain text-to-sql parsing," *arXiv preprint arXiv:2103.02227*, 2021.

[42] V. Zhong, M. Lewis, S. I. Wang, and L. Zettlemoyer, "Grounded adaptation for zero-shot executable semantic parsing," *arXiv preprint arXiv:2009.07396*, 2020.

[43] W. Yang, P. Xu, and Y. Cao, "Hierarchical neural data synthesis for semantic parsing," *arXiv preprint arXiv:2112.02212*, 2021.

[44] T. Yu, C.-S. Wu, X. V. Lin, B. Wang, Y. C. Tan, X. Yang, D. Radev, R. Socher, and C. Xiong, "Grappa: Grammar-augmented pre-training for table semantic parsing," *arXiv preprint arXiv:2009.13845*, 2020.

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

[46] J. Yang, B. Hui, M. Yang, J. Yang, J. Lin, and C. Zhou, "Synthesizing text-to-sql data from weak and strong llms," *arXiv preprint arXiv:2408.03256*, 2024.

[47] G. Lee, H. Hwang, S. Bae, Y. Kwon, W. Shin, S. Yang, M. Seo, J.-Y. Kim, and E. Choi, "Ehrsql: A practical text-to-sql benchmark for electronic health records," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 589–15 601, 2022.

[48] Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, A. Yang, D. Liu, J. Lin *et al.*, "Qwen3 embedding: Advancing text embedding and reranking through foundation models," *arXiv preprint arXiv:2506.05176*, 2025.

[49] Y. Gan, X. Chen, and M. Purver, "Exploring underexplored limitations of cross-domain text-to-sql generalization," *arXiv preprint arXiv:2109.05157*, 2021.

[50] Y. Gan, X. Chen, Q. Huang, M. Purver, J. R. Woodward, J. Xie, and P. Huang, "Towards robustness of text-to-sql models against synonym substitution," *arXiv preprint arXiv:2106.01065*, 2021.

[51] X. Deng, A. H. Awadallah, C. Meek, O. Polozov, H. Sun, and M. Richardson, "Structure-grounded pretraining for text-to-sql," *arXiv preprint arXiv:2010.12773*, 2020.

[52] OpenAI, "GPT-4o mini: Advancing Cost-Efficient Intelligence," 2024, accessed: 2025-10-25. [Online]. Available: https://openai.com/zh-Hans-CN/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

[53] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[54] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[55] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[56] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu *et al.*, "Qwen2. 5-coder technical report," *arXiv preprint arXiv:2409.12186*, 2024.

[57] Qwen Team, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, "Qwen2.5 Technical Report," 2024. [Online]. Available: https://arxiv.org/abs/2412.15115

[58] S. Huang, T. Cheng, J. K. Liu, J. Hao, L. Song, Y. Xu, J. Yang, J. Liu, C. Zhang, L. Chai *et al.*, "Opencoder: The open cookbook for top-tier code large language models," *arXiv preprint arXiv:2411.04905*, 2024.

[59] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[60] M. Mishra, M. Stallone, G. Zhang, Y. Shen, A. Prasad, A. M. Soria, M. Merler, P. Selvam, S. Surendran, S. Singh *et al.*, "Granite code models: A family of open foundation models for code intelligence," *arXiv preprint arXiv:2405.04324*, 2024.

[61] IBM, "IBM Granite 3.1: Powerful performance, longer context, new embedding models and more," Dec. 2024, accessed: 2025-10-25. [Online]. Available: https://www.ibm.com/new/announcements/ibm-granite-3-1-powerful-performance-long-context-and-more

[62] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[63] C. Guo, Z. Tian, J. Tang, P. Wang, Z. Wen, K. Yang, and T. Wang, "A case-based reasoning framework for adaptive prompting in cross-domain text-to-sql," *CoRR*, 2023.