OpenSQL: Data-Efficient Text-to-SQL for Open-Source LLMs via Synthesized Intermediate Supervision

Ruilin Hu
Tsinghua University
hrl24@mails.thu.edu.cn

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

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Shuangqiao Wu
Tencent
ivansqwu@tencent.com

Yun Luo
Tencent
cloudluo@tencent.com

ABSTRACT

The Text-to-SQL task enables non-expert users to query structured data through natural language. While recent methods based on closed-source large language models (LLMs) achieve strong performance, their high inference cost, data privacy concerns, and limited transparency hinder real-world deployment. Open-source LLMs are a promising alternative; however, training them for Text-to-SQL remains challenging due to scarce task-specific annotations and the difficulty of learning reliable grounding and reasoning solely from sparse end-to-end supervision. To address these challenges, we present OpenSQL, a data-efficient framework that improves Text-to-SQL performance of open-source LLMs via synthesized intermediate supervision. OpenSQL converts limited (Question, SQL) pairs into rich, task-decomposed training signals that guide the model to learn critical intermediate decisions. Concretely, (1) we train a global–local schema linking module with schema-aware learning to identify and refine relevant tables and columns; (2) we introduce reasoning-enhanced SQL generation, which produces diverse candidates along complementary reasoning paths and selects the best one through stepwise clause-level and semantic-level reasoning; and (3) we design a task-aware data augmentation pipeline that provides the intermediate supervision signals to support the entire training process. With the same 32B LLM backbone, OpenSQL achieves 70.0% accuracy on Bird-dev using only 14K training samples, outperforming the advanced open-source Text-to-SQL model, OmniSQL, which uses 2.5M training samples.

PVLDB Reference Format:
Ruilin Hu, Yuyu Luo, Guoliang Li, Shuangqiao Wu, Yun Luo. OpenSQL: Data-Efficient Text-to-SQL for Open-Source LLMs via Synthesized Intermediate Supervision. PVLDB, 19(7): 1628 - 1642, 2026.
doi:10.14778/3801059.3801074

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

Proceedings of the VLDB Endowment, Vol. 19, No. 7 ISSN 2150-8097.
doi:10.14778/3801059.3801074

1 INTRODUCTION

The Text-to-SQL task translates natural language questions into executable SQL queries over a given database, enabling non-expert users to access and analyze structured data [43, 65, 94]. Recent advances in large language models (LLMs) have significantly improved performance on this task. Existing methods can be broadly categorized into two types: methods based on closed-source LLMs [5, 16, 50] and those leveraging open-source models [30, 32, 52, 55].

Text-to-SQL with Closed-Source LLMs. Recent methods, *e.g.,* CHASE-SQL [50], have shown strong performance on widely used benchmarks like Bird [37] by prompting closed-source LLMs (*e.g.,* Gemini [18], GPT-4o [48]). While effective in academic settings, these approaches face practical limitations in real-world deployment [31]. First, encoding large database schemas into few-shot prompts incurs substantial costs on commercial LLM services. For instance, CHASE-SQL [50] achieves state-of-the-art accuracy but relies on closed-source LLMs such as Gemini-1.5-pro, which has over 200B parameters. CHASE-SQL costs around $0.6 per question, making it prohibitive at an enterprise scale. Second, many production databases involve proprietary or sensitive information (*e.g.,* business logic), making it infeasible to send schemas or questions to external APIs due to privacy and compliance concerns.

Text-to-SQL with Open-Source LLMs. Alternatively, Text-to-SQL solutions using open-source LLMs (*e.g.,* Llama3 [12]) offer advantages such as cost-efficiency, deployment flexibility, and data privacy [43, 78]. With rapid progress in their language understanding and code generation capabilities [68, 93], they present a promising alternative to closed-source LLMs for the Text-to-SQL task [35].

However, making small and affordable open-source LLMs truly competitive with large closed-source LLMs for Text-to-SQL tasks remains challenging. First, compared with frontier closed-source LLMs, smaller open-source LLMs have fewer parameters and weaker instruction-following ability, which restricts their capability to reason over complex database schemas and long contexts [31, 43]. As a result, they often fail at crucial sub-skills such as schema linking and handling compositional SQL structures. Second, public Text-to-SQL datasets are limited in both size and complexity [35]. Even when we have end-to-end (Question, SQL) pairs, directly fitting the final SQL may suffer from sparse and delayed supervision. A small model can easily overfit surface patterns and still fail to robustly learn (i) schema alignment/grounding decisions and (ii) the reasoning behaviors needed to search and select among multiple SQL candidates.

Table 1: Comparison with SOTA Text-to-SQL methods.

Method	Paradigm	Inference Strategy	Training Data	Overall Cost (Training + Inference)
CHESS [67]	Retrieval + Prompting	Execute + Revise	No need for training	High
CHASE [50]	Agentic Prompting Pipeline	Multi-Prompt + Round-Robin Vote	Low	Very High
OmniSQL [33]	End-to-end Data Synthesis + SFT	CoT-based Generation	High (2.5 Million)	Medium
OpenSQL	**Training with Intermediate Supervision + Test-time Compute**	**Syntax-Controlled Generation + Stepwise Selection**	**Low: 14K (Question, SQL) pairs**	**Low**

Our Methodology. To address the above challenges, we propose OpenSQL, a data-efficient framework that enhances open-source LLMs for Text-to-SQL with schema-aware learning and reasoning-enhanced SQL generation, while improving training effectiveness through synthesized intermediate supervision. The **key idea** of OpenSQL is to decompose Text-to-SQL into well-defined stages with learnable objectives and to transform limited end-to-end training data into rich, task-aware intermediate supervision that explicitly guides the model in making critical intermediate decisions.

To realize this idea, OpenSQL integrates three components. First, schema linking is crucial for grounding a question to the database schema, but open-source LLMs often suffer from low recall and overlook critical schema elements, resulting in missed schema elements. To address this, OpenSQL explicitly trains a global-local schema linking module through schema-aware learning. Given a question and a database schema, the global linker identifies a set of relevant tables and associated columns, while the local linker refines this selection by examining unselected columns within the chosen tables. This coarse-to-fine process ensures precise alignment of schema semantics and reduces the likelihood of overlooking critical elements.

Second, open-source LLMs often struggle with complex SQL generation. To address this, OpenSQL employs a reasoning-enhanced SQL generation framework inspired by test-time scaling. During decoding, OpenSQL generates diverse SQL candidates via complementary syntax-guided reasoning paths, enabling a single model to explore multiple problem-solving perspectives. The generated candidates are evaluated by a specialized SQL selector model with stepwise reasoning, which performs clause-level and semantic-level comparisons to identify the best SQL.

Third, to address the data scarcity challenge, we propose a data augmentation pipeline designed for data efficiency through intermediate supervision. Specifically, OpenSQL synthesizes tailored training targets for each specialized module, including precision-oriented labels for schema linking, syntax-guided reasoning paths, and stepwise rationales for SQL selection. This strategy transforms limited end-to-end annotations into rich, fine-grained supervision signals, making OpenSQL a self-sufficient and data-efficient framework that maximizes the utility of available training resources.

As summarized in Table 1, OpenSQL embodies a distinctive paradigm in Text-to-SQL. Unlike prompting-based frameworks such as CHESS [67] and CHASE-SQL [50], which depend on the expensive reasoning capability of strong proprietary LLMs, OpenSQL internalizes these capabilities directly into open-source LLMs through the co-design of task decomposition and learning schemes, creating a self-contained system. Furthermore, distinct from synthesis-based methods like OmniSQL [33] that rely on millions of end-to-end synthesized (Question, SQL) pairs, OpenSQL focuses on intermediate supervision. By synthesizing fine-grained signals for specific subtasks, OpenSQL shifts the training objective from learning output mappings to mastering the underlying reasoning process, thereby achieving better performance with orders of magnitude less data.

Contributions. This paper makes the following contributions:

(1) Data-Efficient Training Pipeline with Synthesized Intermediate Supervision. We systematically decompose the Text-to-SQL task into multiple learnable objectives and design tailored learning strategies to internalize specialized capabilities for each sub-task. To support training open-source LLMs, we design a task-aware data augmentation pipeline, which synthesizes intermediate supervision signals to provide targeted guidance for each sub-module.

(2) OpenSQL Inference Framework with Test-Time Computing. We propose a cohesive inference framework for Text-to-SQL that unifies (i) global-local schema linking to ensure high-recall schema grounding via two-stage retrieval, and (ii) reasoning-enhanced SQL generation to tackle logical complexity via a test-time compute strategy that combines syntax-guided candidate expansion with stepwise reasoning selection to identify the most accurate SQL.

(3) Extensive Experiments. We have conducted extensive experiments across five LLMs and seven datasets. OpenSQL shows strong and robust performance. On Bird-dev dataset, OpenSQL achieves 70.0% execution accuracy with a 32B base LLM, outperforming the state-of-the-art data synthesis-based model (OmniSQL) and closely approaching CHASE-SQL at 74.5%, which relies on the proprietary Gemini-1.5-Pro LLM with over 200B parameters.

2 RELATED WORK

Text-to-SQL with Closed-Source LLMs. Current Text-to-SQL studies primarily leverage proprietary LLMs via advanced prompting and pipeline engineering. Key strategies include decomposing complex queries [51, 56, 88], optimizing few-shot example retrieval [15, 38, 44, 59] and employing multi-stage frameworks [3, 13, 16, 27, 31, 69, 79, 80]. These methods leverage the large parameter scale and strong in-context learning abilities of closed-source LLMs for Text-to-SQL, but they are limited by high overhead and privacy concerns.

Text-to-SQL with Open-Source Models. In the pre-LLM era, prevailing Text-to-SQL methods [4, 14, 21, 34, 36, 54, 62, 70] relied on fine-tuning pretrained models as specialized modules to enhance schema understanding and logical reasoning. In contrast, recent works [19, 35, 42, 45, 52, 53, 87] design training strategies to improve the performance of open-source LLMs in an end-to-end manner. Compared to these methods, OpenSQL features the coordination of fine-grained task decomposition and data synthesis.

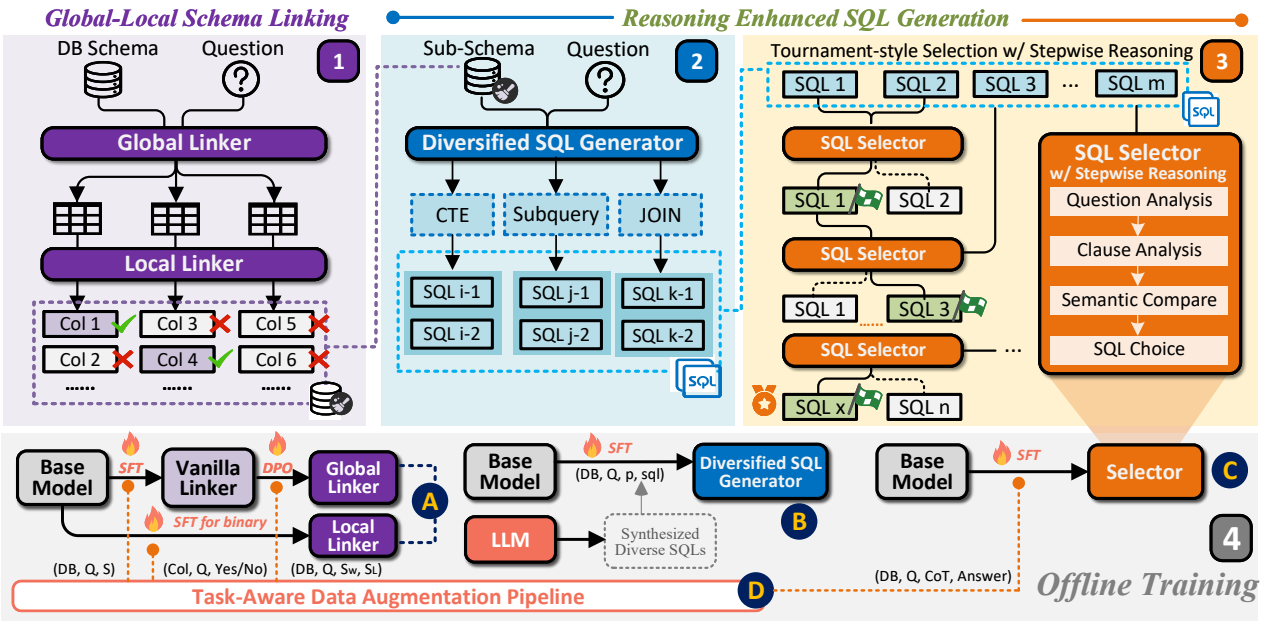

Figure 1: The OpenSQL framework: Given a user question, OpenSQL prunes the schema to reduce problem complexity, explores different reasoning paths to generate SQL candidates, and selects the most appropriate SQL through stepwise reasoning.

LLM Test-time Computing. Several studies [2, 64] have shown that LLMs enhance reasoning capabilities by increasing the test-time compute. A line of work focuses on aggregating diverse reasoning paths [29, 40, 75, 83]. Another paradigm is forcing LLMs to generate thought tokens before they output the result [9, 20, 22, 24, 47, 61, 76, 86]. Our reasoning-enhanced SQL generation framework combines both multiple reasoning paths and thought tokens to achieve controllable test-time computation for Text-to-SQL.

Entity Linking and Schema Matching. Foundational techniques in Entity Linking and Schema Matching are critical for grounding natural language questions to schemas. Classic entity linking studies focus on disambiguating mentions to knowledge base entities via type systems [58] or dense retrieval [77, 81], while schema matching works leverage deep learning to align heterogeneous schema attributes [6, 10, 39, 46, 63]. These concepts and methods underpin the schema linking technique in Text-to-SQL [5, 34, 41, 70, 85].

Data Agents. There are numerous studies on agentic data systems that analyze heterogeneous data, including unstructured, semi-structured, and structured data [71–73, 92]. These systems convert text queries into execution plans composed of relational and semantic operators, which are then optimized and executed [66, 95].

3 OPENSQL OVERVIEW

We introduce OpenSQL, a data-efficient framework designed to improve the Text-to-SQL performance of open-source LLMs. As shown in Figure 1, OpenSQL enhances both training and inference phases through schema-aware learning, reasoning-enhanced SQL generation, and task-aware data augmentation.

3.1 Online Inference Phase

Given a natural language question and a database schema $\mathcal{D}$, OpenSQL performs the following steps during the inference phase.

Global–Local Schema Linking. To reduce reasoning complexity, OpenSQL first prunes the full database schema $\mathcal{D}$ into a question-relevant sub-schema using a *global–local schema linking module*, as shown in Figure 1-❶. The global linker selects coarse-grained relevant tables and columns, while the local linker refines this selection at column-level granularity. This strategy improves the recall for downstream SQL generation.

Diversified SQL Candidate Generation. Using the pruned schema and the natural language question, OpenSQL generates a set of diverse SQL candidates by exploring multiple reasoning paths (*i.e.,* CTE, nested subqueries and normal JOIN), as shown in Figure 1-❷. This step expands the search space by exploring different syntactic and semantic variations of SQL queries. This diversity improves the likelihood that the correct SQL query is included among the candidates, which is crucial for effective downstream selection.

Tournament-style SQL Selection with Stepwise Reasoning. As shown in Figure 1-❸, the generated SQL candidates are then evaluated by a lightweight SQL selector that performs pairwise comparisons guided by stepwise reasoning. For each pair of SQL queries, the selector executes a step-by-step clause-level and semantic-level comparison. To enhance efficiency, a tournament-style selection strategy is employed: in each round, one candidate is eliminated based on the reasoning comparison, and the process continues until the most appropriate SQL query is selected as the final output.

3.2 Offline Training Phase

As shown in Figure 1-❹, OpenSQL improves the performance of open-source LLMs via three complementary training stages, each addressing a key aspect of Text-to-SQL: schema-aware training, diversified SQL generation, and fine-grained SQL selection.

Schema-Aware Training. To enhance the model's understanding of database structures, OpenSQL explicitly trains the global–local schema linker. The global linker is optimized through supervised fine-tuning and preference learning (*e.g.,* DPO [57]) to select relevant tables, while the local linker is trained as a binary classifier to align question spans with specific columns. This structured supervision improves recall in schema element selection while keeping high precision, and stands as a prerequisite for SQL generation.

Training Reasoning-Enhanced SQL Generation Framework. To enable robust inference-time reasoning, OpenSQL trains two modules as part of the reasoning-enhanced SQL generation framework: the diversified SQL generator and the stepwise SQL selector.

First, OpenSQL trains the diversified SQL generator to explore multiple reasoning paths, thereby expanding the search space and increasing the likelihood of generating the correct SQL query. This is achieved by introducing special control tokens and training on SQL queries that conform to specific syntactic structures.

Second, OpenSQL trains the SQL selector to choose the best query from the top-k SQL candidates. The training includes pairwise comparisons of SQL queries, with the selector performing clause-level and semantic-level analysis to identify the most accurate query. The selector is trained using synthetic data with stepwise reasoning paths, which guide the model to analyze the question and the SQL queries, comparing them on structural and semantic differences.

3.3 Task-Aware Data Augmentation

To alleviate the scarcity of high-quality annotated data, OpenSQL adopts a unified synthetic data generation pipeline that supports all learning components. Specifically, we generate four categories of training data for: global schema linking, local schema linking, SQL candidate generation, and fine-grained SQL selection modules. This comprehensive supervision maximizes the utility of training data.

4 GLOBAL-LOCAL SCHEMA LINKING

In this section, we first present the design of our global-local schema linking module (Section 4.1), followed by a detailed explanation of our schema-aware training framework (Section 4.2).

4.1 Global-Local Schema Linking Overview

Schema linking is an essential component in Text-to-SQL solutions, bridging natural language questions with the database schema. However, existing schema linking approaches face challenges in achieving a balance between precision and recall. A common issue is that overly strict schema pruning (*i.e.,* selecting only the most relevant tables and columns) can lead to information loss, where important schema components are overlooked, resulting in inaccurate SQL generation. On the other hand, when the entire schema is inputted, it causes inefficiencies in large databases, especially when using open-source models that struggle with longer inputs.

To address these challenges, we propose Global-Local Schema Linking, a two-stage strategy that performs schema selection at both the global (*i.e.,* selecting tables and columns) and local (*i.e.,* focusing on unselected columns within the already-chosen tables) levels. This approach enables efficient schema pruning while ensuring the necessary schema components are retained for SQL generation.

Algorithm 1: Global-Local Schema Linking

Input: Global schema linker M_{global}, local schema linker M_{local}, database schema $\mathcal{S}$, natural language question q.
Output: Pruned database schema $\mathcal{S}_{pruned}$.

1 $\mathcal{S}_{coarse} \leftarrow M_{global}(\mathcal{S}, q)$ // Global schema linking
2 $\mathcal{S}_{pruned} \leftarrow \mathcal{S}_{coarse}$
// For each table chosen by global schema linking
3 **foreach** table $T \in \mathcal{S}_{coarse}$
// For each column in T not included in $\mathcal{S}_{coarse}$
4 $C_{unselected} \leftarrow$ GetUnselctedColumns$(\mathcal{S}_{coarse}, T)$
5 **foreach** column $c \in C_{unselected}$ // Local schema linking
6 **if** $M_{local}(c, q) ==$ True
7 $\mathcal{S}_{pruned} \leftarrow \mathcal{S}_{pruned} \cup \{c\}$
8 **return** $\mathcal{S}_{pruned}$

Global Schema Linker. The global schema linking model M_{global} is a high-capacity model trained to perform a coarse-grained selection of tables and columns from the full database schema. In particular, M_{global} is responsible for identifying all tables necessary for constructing JOIN clauses in cases where multi-table queries are needed. However, since M_{global} focuses on high-level table selection to form abstract entities capable of answering the question, it often overlooks subtle cues in the question that indicate the need for specific columns. Therefore, a column-level complement for schema linking is required.

Local Schema Linker. The local schema linking model M_{local} aims to capture connections between the user question and individual columns. Specifically, M_{local} acts as a binary classifier: Given a question and a column description, M_{local} attends to subtle cues in the question and determines whether the given column is relevant to answer it. Since M_{local} needs to evaluate multiple columns for each question, we configure it as a low-cost small language model.

Global-Local Schema Linking Strategy. Algorithm 1 illustrates our global-local schema linking strategy. Given the question and the full database schema, we first use M_{global} to perform coarse-grained global schema linking (Line 1) to preliminarily select needed tables and columns. After that, within the scope of the tables selected by M_{global} (Line 3), M_{local} performs local schema linking, where M_{local} further examines unselected columns (Line 5) to determine their relevance to the user question (Line 6), and adds relevant columns to the predicted sub-schema (Line 7). Our Global-Local Schema Linking strategy addresses a challenge in schema linking by balancing precision and recall in selecting schema components. It ensures the detailed question semantics match the relevant columns while selecting the sub-schema structure, thereby reducing errors caused by schema linking and enhancing the end-to-end accuracy.

4.2 Schema-Aware Learning

As discussed in Section 4.1, schema linking faces the challenge of improving recall while minimizing precision loss, particularly with complex large databases. To address this, we introduce schema-aware learning, a joint strategy that optimizes both global-level and local-level schema selection. First, we design *schema-aware supervised fine-tuning* for both global and local schema linking. This trains the model to predict relevant schema components using

labeled data, ensuring accurate mappings between natural language questions and database elements. Next, *schema-aware preference learning* is applied to train the global linker. This method refines its selection by incorporating preference signals, helping the model favor the sub-schema that contains sufficient database elements to answer the question. Together, these two approaches effectively enhance recall while maintaining high precision.

Schema-Aware Supervised Fine-tuning. Given a natural language question $q \in \mathcal{Q}$ ($\mathcal{Q}$ is the set of all valid user questions) and full database schema $\mathcal{D}$, we first perform supervised fine-tuning to train the global linker M_{global} to predict the required sub-schema. We denote this process as learning a mapping function:

$$M_{global} : \mathcal{Q} \times \mathcal{D} \rightarrow \mathcal{P}(\mathcal{D}), \quad \text{where } M_{global}(q, \mathcal{D}) = \mathcal{D}^* \quad (1)$$

where $\mathcal{P}(\mathcal{D})$ is the powerset of $\mathcal{D}$, and $\mathcal{D}^*$ is a subset of $\mathcal{D}$ containing the **exact** the tables and columns needed to answer q.

In addition to M_{global}, we train M_{local} with supervised fine-tuning to capture the subtle relationship between natural language questions and individual columns. We denote this process as learning a mapping function:

$$M_{local} : \mathcal{Q} \times C \rightarrow \{0, 1\} \quad (2)$$

where C is the set of columns in databases. Each column in C is represented by its description, and exemplar values. M_{local} returns 1 if the column is relevant to the user's question, and 0 otherwise.

Schema-Aware Preference Learning. After supervised fine-tuning, M_{global} is capable of predicting the required sub-schema to answer the question. However, limited model capacity often results in the model overlooking tables and columns during the schema linking stage. Inspired by [49], which fine-tunes LLMs to avoid undesired behaviors via preference learning, we adopt preference learning to the global schema linking model M_{global} to instill in the model a bias that any missing selection is unacceptable. Specifically, when the model is uncertain about whether to select a database element (a table or a column), it should favor retaining the element rather than discarding it, thereby ensuring higher recall and minimizing the risk of excluding relevant database elements.

Given a question q and two schema subsets $\mathcal{D}^+$ (preferred) and $\mathcal{D}^-$ (not preferred), where $\mathcal{D}^+$ includes all necessary database elements and $\mathcal{D}^-$ omits one or more relevant elements, we use direct preference optimization [57] to further enhance **the global linker** to favor $\mathcal{D}^+$ over $\mathcal{D}^-$, by maxmizing the following objective:

$$\mathbb{E}\left[\log \sigma\left(\beta \log \frac{p_\theta(\mathcal{D}^+ \mid q, \mathcal{D})}{p_{\text{ref}}(\mathcal{D}^+ \mid q, \mathcal{D})} - \beta \log \frac{p_\theta(\mathcal{D}^- \mid q, \mathcal{D})}{p_{\text{ref}}(\mathcal{D}^- \mid q, \mathcal{D})}\right)\right] \quad (3)$$

where θ is the parameter set of the global linker, β is a hyperparameter. $p_\theta(\mathcal{D}^+ \mid q, \mathcal{D})$ and $p_\theta(\mathcal{D}^- \mid q, \mathcal{D})$ are the probabilities assigned by the global linker to the schema subsets $\mathcal{D}^+$ and $\mathcal{D}^-$, conditioned on the question q and full schema $\mathcal{D}$. And p_{ref} represents the corresponding probabilities assigned by a fixed reference global linker that has been trained using only supervised fine-tuning. With this additional preference learning process, M_{global} is encouraged to retain uncertain database elements, enhancing the robustness of our global-local schema linking module.

Question: How many action games are there in total?

CTE : **Progressive reasoning through `WITH...AS`**

```
WITH action_genre AS
(SELECT id FROM genre WHERE g_name='Action')
SELECT COUNT(T1.id) FROM game AS T1
JOIN action_genre AS T2 ON T1.genre_id = T2.id;
```

SUBQ : **Hierarchical reasoning through subquery**

```
SELECT COUNT(id) FROM game WHERE genre_id IN
(SELECT id FROM genre WHERE g_name = 'Action');
```

NORMAL : **Direct reasoning through `JOIN`**

```
SELECT COUNT(T1.id) FROM game AS T1 JOIN genre AS T2
ON T1.genre_id = T2.id WHERE T2.g_name = 'Action';
```

Figure 2: Example of multiple reasoning paths.

5 REASONING-ENHANCED SQL GENERATION

In this section, we will first outline our reasoning-enhanced SQL generation framework (Section 5.1). Then, we detail its two components: diversified SQL candidate generation (Section 5.2) and tournament-style SQL selection with stepwise reasoning (Section 5.3).

5.1 Overall SQL Generation Framework

Open-source LLMs struggle with accurate Text-to-SQL reasoning over complex databases due to the small parameter size and limited instruction-following ability. From preliminary experiments, we observed a notable improvement in pass@k Text-to-SQL accuracy over pass@1 accuracy, suggesting that open-source LLMs actually possess the capability to generate correct queries and can benefit from multiple decoding attempts in complex Text-to-SQL tasks.

Motivated by this, we propose a general two-stage inference framework to enhance the reasoning capabilities of open-source models in Text-to-SQL. In the first stage **(Expansion)**, the model generates multiple SQL candidates for the question. The goal of this stage is to expand the search space, ensuring that the correct SQL query is included among the generated candidates. In the second stage **(Convergence)**, the model performs a careful selection process to choose the best SQL from the candidates, which will be chosen as the final answer. In this framework, models only focus on covering the correct SQL in the Expansion phase and selecting the best SQL in the Convergence phase. As a result, the Text-to-SQL system can bypass steps such as SQL fixing [7] or refinement [5].

5.2 Diversified SQL Candidate Generation

Although open-source LLMs are capable of generating correct SQL queries, they often fail to produce the correct result in a single attempt. Therefore, we prioritize covering the correct SQL with multiple SQL candidates. To this end, we introduce a diversified SQL generator that produces multiple candidate queries through complementary reasoning paths, where a reasoning path represents *a specific strategy* for translating natural language into SQL. Specifically, we design three such reasoning paths, each based on a different SQL syntactic structure. During decoding, the model

follows each path to generate candidate queries, then all generated candidates are jointly collected for downstream selection.

SQL Syntax-based Reasoning Paths. A question can be answered by multiple SQL queries that follow different reasoning paths. Motivated by this, we propose to generate SQL candidates via distinct reasoning paths so that the generated queries complement each other and collectively increase the likelihood of producing the correct SQL. We categorize three reasoning paths based on SQL syntax, where the main distinction lies in how multiple tables are connected and intermediate results are organized to answer the question:

- **Common Table Expression (CTE).** Queries in this category employ the `WITH...AS` syntax. They represent a reasoning process in which the final solutions are progressively derived through consecutive intermediate results.
- **Subquery.** Queries in this category employ nested subquery. They represent a reasoning process that embeds auxiliary queries within a main query to filter, aggregate, or compute intermediate results before getting the answer.
- **Normal.** Queries in this category do not involve `CTE` or nested subqueries. They mainly consist of single-table queries or multi-table queries using `JOIN`, representing a direct approach that solves the problem with basic SQL constructs.

In essence, these three reasoning paths represent high-level strategies for problem-solving, while the SQL syntax (`CTE`, Subquery, and Normal forms) serves as the concrete classification criteria. Together, these three reasoning paths capture complementary strategies for generating SQL and can provide clear guidance for directing the model to produce diverse SQL candidates.

Diversified SQL Generator. With our reasoning path definitions, we trained an SQL generator to produce multiple SQL queries along different reasoning paths at inference time. We achieved this by introducing three special control tokens: [CTE], [SUBQ], [NORMAL], where each control token is associated with a specific reasoning path, as shown in Figure 2. These control tokens serve as instructions for the SQL generation model. During SQL generation, these control tokens are appended to the input prompt. The control token guides the model to follow the corresponding reasoning path and generates SQL queries that conform to the associated structure. Each decoding process for SQL candidate generation is formulated as:

$$M_{gen}(q, \mathcal{D}^*, p) = y \tag{4}$$

where $q \in \mathcal{Q}$ denotes the user question, $\mathcal{D}^*$ is the sub-schema and $p \in \{$[CTE], [SUBQ], [NORMAL]$\}$ is the reasoning path control token. With each control token, the model generates an SQL query y that adheres to the specified reasoning path. If the model determines that a reasoning path is not suitable for the question, it can output a [N/A] token to indicate abstention. Finally, the SQL queries generated from all reasoning paths are then aggregated and provided as input to the SQL selection stage. Compared to generating one SQL, our diversified SQL generator leverages the syntactic flexibility of SQL to cover diverse reasoning paths efficiently with a single model.

```
Question: Who is the oldest patient with the highest total cholesterol
  (T-CHO). State the patient ID and T-CHO index.
SQL1: SELECT T1.ID, T2.`T-CHO` FROM Patient AS T1 INNER JOIN Lab AS T2
   ON T1.ID = T2.ID ORDER BY T2.`T-CHO` DESC, T1.Birthday ASC LIMIT 1;
SQL2: SELECT T1.ID, T2.`T-CHO` FROM Patient AS T1 INNER JOIN Lab AS T2
   ON T1.ID = T2.ID ORDER BY T1.Birthday ASC, T2.`T-CHO` DESC LIMIT 1;
SQL Selector Output:
1. Question Analysis.
The user wants to identify the oldest patient who has the highest total
  cholesterol. The result should include the patient ID and the
  corresponding T-CHO value.
2. SQL 1 Analysis.
- Join Logic: SQL 1 performs an 'INNER JOIN' between the 'Patient' and
  'Lab' tables using the 'ID' column, which is correct since it links
  patients with their laboratory test results.
- Sorting Logic: The results are first sorted by T-CHO in descending
  order to prioritize higher values, and then by Birthday in ascending
  order to ensure that, among identical T-CHO values, the oldest patient
  appears first.
3. SQL 2 Analysis.
- Join Logic: SQL 2 performs an INNER JOIN between the Patient and
  Laboratory tables using the ID column, which is correct.
- Sorting Logic: The results are first sorted by Birthday in ascending
  order to prioritize the oldest patients, and then by T-CHO in
  descending order to ensure that among the oldest patients, the one
  with the highest T-CHO appears first.
4. Difference Comparison.
- Primary Sorting: SQL 1 Orders by 'T-CHO' DESC first, ensuring the
  highest 'T-CHO' appears first. SQL2 Orders by 'Birthday' first,
  ensuring the oldest patient appears first.
- Secondary Sorting: SQL 1 Orders by 'Birthday' second, ensuring the
  oldest patient is chosen. SQL2 Orders by 'T-CHO' DESC second, ensuring
  the highest 'T-CHO' is chosen.
- Focus on Oldest Patient: SQL1 Focuses on the highest 'T-CHO' first,
  then selects the oldest patient. SQL2 Focuses on the oldest patient
  first, then selects the highest 'T-CHO'.
5. SQL Choice: SQL1 is correct.
```

Figure 3: An example of stepwise SQL comparison.

5.3 Tournament-Style SQL Selection with Stepwise Reasoning

The selection module aims to identify the best query from the SQL candidates. While self-consistency can guide selection [75], prior work [50] has shown that it results in suboptimal accuracy. Directly training a model to rerank SQL candidates also faces challenges, as the subtlety of semantic errors in SQL makes models prone to overfitting on shallow patterns rather than learning to distinguish between correct and incorrect queries. These limitations call for an SQL selection module capable of capturing subtle semantic differences in SQL queries and selecting the one that best aligns with the user's intent. To overcome these limitations, we propose a stepwise SQL selector and a tournament-style selection strategy.

Pairwise Comparisons with Stepwise Reasoning. To select the best SQL query from multiple candidates, we reduce the task to *pairwise comparisons* between SQL candidates. This stepwise reasoning process evaluates each pair based on syntax and semantic differences. As shown in Figure 3, each pairwise comparison is formulated as a generation-and-reasoning task, which is carried out by our SQL selector. Given a pair of SQL candidates, the selector initially produces a detailed reasoning path and then determines which query better reflects the user's intent. Specifically, the selector uses a four-step reasoning process to compare SQL candidates:

- **Question Analysis**. Restate the user's intent and think about which columns are the output.
- **Clause Analysis**. Analyze the clauses in the two SQL candidates. Since SQL queries are naturally composed of multiple clauses, this step conducts a sequential analysis of each clause. It checks

for syntax errors and serves as an intermediate process to support deeper semantic analysis.

- **Semantic Comparison**. Compare the two SQL queries on the semantic level. This step focuses on understanding the overall meaning of each query, examining how the queries align with user requirements, and identifying logical errors.
- **SQL Choice**. Make a final decision on which SQL query is preferred based on the reasoning results.

This SQL-specialized process leverages a structured form of chain-of-thought reasoning and test-time compute to compare SQL queries. Through clause-level inspection within SQL queries and semantic-level comparison across SQL candidates, it enables a systematic evaluation of SQL pairs. Compared to direct reasoning, this structured process also mitigates the position bias issue, which is an inherent limitation of the LLM-as-a-Judge paradigm [91].

Tournament-Style SQL Selection. To select the best SQL query from a set of candidates, we propose a lightweight tournament-style strategy. As illustrated in Figure 1-❸, given m generated SQL candidates $\{y_i\}_{i=1}^{m}$ at test time, we perform the stepwise comparison between each SQL candidate pair and eliminate suboptimal candidates at each step. The final remaining SQL candidate is selected as the output. Since each comparison eliminates one SQL candidate, the actual number of pairwise comparisons performed is fewer than $m-1$. Although each pairwise comparison with stepwise reasoning incurs additional latency, our tournament-style selection strategy effectively mitigates the overall inference latency by reducing the number of required comparisons.

5.4 Training the SQL Reasoning Module

Training the Diversified SQL Generator. To enable the SQL generator to produce SQL queries using multiple reasoning paths during inference, we supervised fine-tune it to learn from SQL queries conditioned by special control tokens. This process can be formulated as learning the following mapping function:

$$M_{gen} : \mathcal{Q} \times \mathcal{D}^* \times \mathcal{P} \rightarrow \mathcal{S} \cup \{\boxed{\text{N/A}}\}, \mathcal{P} = \{\boxed{\text{CTE}}, \boxed{\text{Subq}}, \boxed{\text{Normal}}\} \quad (5)$$

Here, $\mathcal{D}^*$ is the sub-schema produced by the global-local schema linking module, $\mathcal{S}$ is the space of all valid SQL queries and $\mathcal{P}$ is the set of all special control tokens.

More specifically, given a natural language question $q \in \mathcal{Q}$ and a sub-schema $\mathcal{D}^*$, we append each reasoning path control token to the end of q to form the input sequence, and use the corresponding SQL query with the specified syntactic structure as the output sequence. If there is no suitable SQL query for a given reasoning path, the model learns to output an $\boxed{\text{N/A}}$ token. We obtain reliable SQL query annotations with different syntactic structures through the data augmentation module in Section 6. By training the SQL generator with diverse SQL forms, the model can generate SQL queries via multiple reasoning paths and actively avoid generating low-quality SQL when a particular reasoning path is unsuitable.

Training the SQL Selector. Our stepwise SQL selector leverages structured reasoning to make **pairwise** comparisons. The training process is equivalent to learning a binary classification function:

$$M_{selector} : \mathcal{Q} \times \mathcal{D}^* \times \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{R} \times \{z \mid z \in \{0, 1\}\} \quad (6)$$

Algorithm 2: Preference Learning Dataset Construction

Input: Global schema linking dataset $\mathcal{L}_{sl}$, frequency K
Output: Preference dataset $\mathcal{L}_{dpo}$

$\mathcal{L}_{dpo} \leftarrow \emptyset$
foreach $(\mathcal{D}, q \mapsto \mathcal{D}^*) \in \mathcal{L}_{sl}$
  $\mathcal{D}^+ \leftarrow \mathcal{D}^*$ // Set preferred schema as the correct sub-schema
  for $i \leftarrow 1$ **to** K // Generate by random column deletion
    $C \leftarrow$ `RandomlySampleColumns`$(\mathcal{D}^*)$
    $\mathcal{D}^- \leftarrow \mathcal{D}^* \setminus C$
    $\mathcal{L}_{dpo} \leftarrow \mathcal{L}_{dpo} \cup (\mathcal{D}, q \mapsto \mathcal{D}^+, \mathcal{D}^-)$
  foreach `table` $T \in \mathcal{D}^*$ // Generate by table deletion
    $\mathcal{D}^- \leftarrow \mathcal{D}^* \setminus \{T\}$
    $\mathcal{L}_{dpo} \leftarrow \mathcal{L}_{dpo} \cup (\mathcal{D}, q \mapsto \mathcal{D}^+, \mathcal{D}^-)$
return $\mathcal{L}_{dpo}$

where $\mathcal{S}$ is all valid SQL queries, $\mathcal{R}$ represents all stepwise chain-of-thoughts. The comparison label z is 0 if the selector prefers the first SQL query in the pairwise comparison and 1 otherwise.

In detail, we employ supervised fine-tuning to train our pairwise SQL selector. We augment the training data (Section 6) by attaching a chain-of-thought rationale to each pairwise comparison instance. During training, the model is supervised to generate both the rationale and the comparison label.

6 TASK-AWARE DATA AUGMENTATION

In this section, we first outline of our data augmentation framework (Section 6.1), followed by the details of its application to schema linking (Section 6.2) and reasoning-enhanced SQL generation (Section 6.3). The experimental setting is described in Section 7.1.3.

6.1 Overview

As highlighted by [35], data scarcity remains a key challenge in Text-to-SQL, which comprises multiple subtasks requiring different types of supervision. To address this, we propose a task-aware data generation pipeline that systematically augments human-annotated data across these subtasks, ensuring comprehensive model training.

Formally, each human-annotated training instance is represented as $(\mathcal{D}, q) \mapsto y$, where $(\mathcal{D}, q)$ denotes the input database schema and natural language question, and y is the ground-truth SQL. Starting from this limited supervision, our pipeline integrates information extraction, LLM-based data synthesis, and rejection sampling to ensure the quality and task relevance of the generated data.

6.2 Data Augmentation for Schema Linking

Supervised Fine-tuning for Global Schema Linking. The training data for the global linker in this step is formulated as $(\mathcal{D}, q) \mapsto \mathcal{D}^*$, where $\mathcal{D}^*$ is the exact sub-schema corresponding to the correct SQL query y. For each y, we extract all referenced tables and columns in the SQL query to construct $\mathcal{D}^*$. To ensure integrity, we add the primary key of each referenced table and foreign keys required to connect the involved tables into $\mathcal{D}^*$.

Supervised Fine-tuning for Local Schema Linking. The training data for the local linker is formulated as $(t, a, q) \mapsto \{0, 1\}$, where t is the description of an individual table, a is the description of an

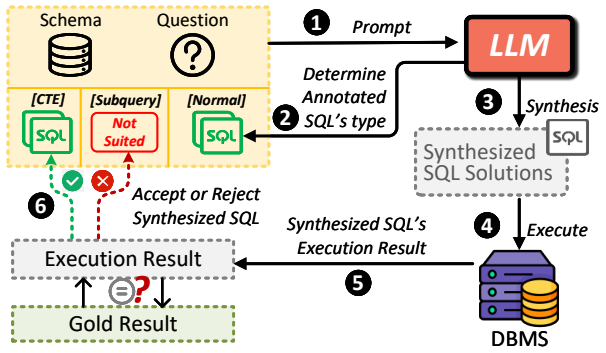

Figure 4: The data augmentation pipeline for SQL generator.

attribute (*i.e.,* column) inside t. For each y in the training instances, we iterate over all columns in the referenced tables and label each column as 1 if it appears in the corresponding sub-schema $\mathcal{D}^*$, and 0 otherwise. These labeled examples are used to construct the training data for local schema linking.

Preference Learning for Global Schema Linking. In this step, for each input $(\mathcal{D}, q)$, the global linker is presented with a sub-schema pair $(\mathcal{D}^+, \mathcal{D}^-)$, and is trained to satisfy the preference constraint that $\mathcal{D}^+$ is more preferred than D^-. Given q and the corresponding sub-schema $\mathcal{D}^*$, $\mathcal{D}^+$ satisfies $\mathcal{D}^* \subseteq \mathcal{D}^+$ and $\mathcal{D}^-$ satisfies $\mathcal{D}^- \subset \mathcal{D}^*$. We use Algorithm 2 to construct the training data. For each training instance (Line 2), we start from the exact referenced sub-schema $\mathcal{D}^*$ (Line 3), remove columns (Line 4) or remove each table (Line 8) to obtain the not-preferred sub-schema $\mathcal{D}^-$. Note that we keep the preferred sub-schema $\mathcal{D}^+$ the same as $\mathcal{D}^*$ to avoid introducing noise.

6.3 SQL Generator and Selector Augmentation

LLM-based Synthesis for Diversified SQL Generator. The training data for the SQL generator is formulated as $(\mathcal{D}^*, q, p \mapsto y)$, where each y is the answer for q and adheres to the control token p. As shown in Figure 4, we designed an LLM-based pipeline to synthesize SQL queries for training. In our data augmentation pipeline, we start with a single manually annotated SQL query. For each training instance, we input the question along with the annotated query into an LLM (Figure 4-❶) and prompt it in one-shot manner to decide the reasoning path of the annotated query (Figure 4-❷) and generate SQL queries in other two reasoning paths (Figure 4-❸). The LLM may abstain to a reasoning path if that path is unsuitable for the question. The generated queries are executed in DBMS (Figure 4-❹) to obtain execution results (Figure 4-❺). If the execution result of a new query matches the correct result, the new SQL query is added. Otherwise, we consider the corresponding reasoning path unsuitable for the question.(Figure 4-❻).

Stepwise Reasoning Annotation for SQL Selector. Each training instance for the stepwise SQL selector can be formulated as $(\mathcal{D}^*, q, y^{correct}, y^{wrong} \mapsto r, z)$, where r is the stepwise reasoning rationale and $z \in \{0, 1\}$ indicates the pair-wise comparison result. The data synthesis process for the SQL selector is outlined in Algorithm 3. We first sample multiple SQL queries for each question

Algorithm 3: Stepwise Reasoning Annotation

Input: Text-to-SQL dataset $\mathcal{L}$, pre-trained LLM $M_{pretrain}$, annotator LLM $M_{annotator}$, sample frequency K_s
Output: Pair-wise SQL comparison dataset $\mathcal{L}_{rationale}$
// For each question q, sample multiple SQL queries
1 $\mathcal{L}_{draft} \leftarrow$ `RunDataset`$(M_{pretrain}, \mathcal{L}, K_s)$
// Judge correctness of the sampled queries using gold SQL
// $\mathcal{L}_{non-rationale} = \{(\mathcal{D}^*, q, y^{correct}, y^{wrong})\}$
2 $\mathcal{L}_{non-rationale} \leftarrow$ `JudgeCorrectness`$(\mathcal{L}_{draft})$
// Annotate the chain-of-thought rationale using LLM
// $\mathcal{L}_{rationale} = \{(\mathcal{D}^*, q, y^{correct}, y^{wrong} \mapsto r, z)\}$
3 $\mathcal{L}_{rationale} \leftarrow$ `Annotate`$(M_{annotator}, \mathcal{L}_{non-rationale})$
// Only use rationale with the correct comparison result
4 $\mathcal{L}_{rationale} \leftarrow$ `FilterCorrect`$(\mathcal{L}_{rationale})$
5 **return** $\mathcal{L}_{rationale}$

with a pretrained model (Line 1). This step allows us to obtain the correct and incorrect SQL queries for each question. After this, we judge the correctness of the sampled queries by comparing their execution results in the DBMS against the gold SQL (Line 2) and leverage an annotator LLM to generate the stepwise reasoning rationale (Line 3). Finally, we filter out only the stepwise reasoning rationales that lead to correct results as the training data (Line 4).

7 EXPERIMENTS

7.1 Experimental Setup

7.1.1 Datasets. We experiment on seven Text-to-SQL benchmarks:

Challenge Benchmark: BIRD [37] is a challenging benchmark containing 12751 annotated (Question, SQL) pairs across 95 databases. BIRD is substantially complex because it contains massive dirty database contents and requires reasoning over external knowledge. We use the BIRD-dev set for evaluation, which contains 1534 pairs.

General Benchmark: SPIDER [84] is a widely used Text-to-SQL benchmark spanning over 138 domains. SPIDER contains 8659, 1034, and 2147 pairs in the train, development, and test sets, respectively.

Robust Benchmark: DR.SPIDER [8] is a variant derived from SPIDER, integrating 17 types of variation across questions, databases, and SQL queries. DR.SPIDER contains more than 15000 datapoints and serves as an extensive benchmark for comprehensively assessing the generalization capabilities of Text-to-SQL systems.

Real-World Benchmarks: KAGGLEDBQA [26], **MIMIC-SQL** [74], **SCIENCE** [90], **SPIDER2** [28] are real-world Text-to-SQL benchmarks for data analysis, healthcare, scientific research, and enterprise business intelligence. For KAGGLEDBQA and MIMIC-SQL, we use the test set. For SCIENCE, we use the development set. For **SPIDER2**, we use the SQLite subset (referring to **SPIDER2-SQLITE**). The four datasets contain 185, 1000, 299, and 135 questions, respectively.

7.1.2 Metric. We consider the prevalent metric: **execution accuracy (EX)**, which evaluates whether the generated SQL extracts the same results as the ground-truth SQL query from the database.

7.1.3 Data Details. We used the training sets from the public Spider and Bird datasets as base training data for our data augmentation pipeline. We used DeepSeek-R1 as the annotation LLM for a balance between cost and quality in Algorithm 2 and Algorithm 3. Before augmentation, we used DeepSeek-R1 to filter noised training data and obtained 6.3K and 7.6K instances from Bird and Spider.

For the global linker, we sampled 70% of the data for SFT and used the remaining to build DPO data with Algorithm 2, where the frequency K was 3. We sampled 16 outputs at 0.8 temperature for each question with OmniSQL-7B as $M_{pretrain}$ in Algorithm 3. All data filtering and augmentation cost 55$. Following [17], the input schemas for the generator and selector consist of the exact used tables and columns augmented with 0–2 randomly added irrelevant tables and 0–2 irrelevant columns per table. For global schema linking, we obtained 9K and 16K datapoints for SFT and DPO. For local schema linking, we sampled positive and negative columns at a 1:1 ratio and obtained 60K binary classification data. For SQL generation and selection, we obtained 29K and 9K SFT data.

7.1.4 Implementation Details. We conducted experiments on a server running Ubuntu 24.04 with an Intel Xeon 8558 CPU, eight NVIDIA H200 GPUs. We fine-tuned open-sourced LLMs to obtain schema linker, SQL generator, and SQL selector models. We conducted experiments on different model architectures (Qwen-2.5-Coder [23], Llama3.1 [12], and OmniSQL [33]). The parameter sizes of LLMs spanned from 7B to 32B. We consistently used the local schema linker fine-tuned from Qwen2.5-Coder-3B.

Training. Our models were trained with the TRL library [1]. The global linker was first trained with SFT, then trained with DPO. The SQL generator and selector were trained with SFT. We trained each model for 3 epochs by default. For the SQL selector, we trained 4 epochs. The SFT learning rate was 1×10^{-5} for Qwen and OmniSQL. For Llama, we halved the SFT learning rate. In DPO, we set beta to 0.2, learning rate to 1×10^{-6}, epoch to 1, and we added an SFT loss with a weight of 0.3. The gradient accumulation steps were 16.

Evaluation. We built vector indexes for database content of `TEXT` types with FAISS [11] and gte-large-en-v1.5 [89]. We used vLLM [25] for inference. Three values with the highest similarity to the question were extracted as examples. For each reasoning path in the SQL generation module, we sampled 8 outputs at 1.5 temperature.

7.1.5 Baselines. We consider published state-of-the-art methods as our baselines, which can be categorized into two categories.

Prompting Methods with Closed-source LLMs. This type of method integrates advanced pipelines and prompt engineering.

Text-to-SQL with Open-source Models. Methods of this type fine-tune models with specific data and potentially employ multiple models. For open-source model baselines, we sampled 24 candidates with 0.8 temperature and applied the self-consistency strategy.

7.2 Overall Performance

Exp-1: How does OpenSQL perform compared to state-of-the-art solutions that based on closed-source LLMs? To evaluate the effectiveness of OpenSQL in improving Text-to-SQL accuracy, we carried out experiments on the Bird and Spider datasets. We applied the data augmentation pipeline to the training sets of Bird and Spider and fine-tuned models with the augmented data, then evaluated OpenSQL on Bird-dev, Spider-dev, and Spider-test datasets. The experimental results are shown in Table 2.

With Qwen2.5-Coder-7B, OpenSQL achieves 67.2% EX on Bird-dev, outperforming all fine-tuning-based baselines of similar parameter scales. With Llama3.1-8B, OpenSQL achieves 65.1% EX on Bird-dev, demonstrating the versatility of OpenSQL across different LLM architectures. Additionally, OpenSQL-32B achieves 70.0% EX on Bird-dev, 88.6% EX on Spider-dev, and 88.3% EX on Spider-test, showing competitive performance with leading methods based on closed-source LLMs. These results indicate that OpenSQL maintains effectiveness across different parameter scales. This experiment demonstrates that OpenSQL effectively enhances the accuracy of open-source LLMs for Text-to-SQL, narrowing the gap with state-of-the-art closed-source LLMs. Notably, while performance scales with model size on the complex Bird benchmark, smaller models occasionally surpass larger counterparts. This is attributed to annotation bias, where stronger models tend to generate more complete SQL formulations (*e.g.,* handling ties), which can be reasonable but mismatched with the single ground-truth execution and thus counted as errors.

Exp-2: How well does OpenSQL generalize to unseen user needs, changing schemas and databases of different dialects? To evaluate the robustness of OpenSQL when applied to unseen data, we first conducted experiments on Dr.spider, which contains perturbed databases, questions, and SQL queries from Spider. We compared OpenSQL with three baselines: the published state-of-the-art on Dr.spider (ZeroNL2SQL+GPT-4) and two top open-source models from Bird (ExCoT+Qwen2.5-Coder-32B and OmniSQL-32B). For open-source models, we sampled 24 outputs at 0.8 temperature and voted with self-consistency. As shown in Table 3, OpenSQL + Qwen2.5-Coder-7B and OpenSQL + OmniSQL-7B both averagely achieve 79.8% EX, surpassing ZeroNL2SQL+GPT-4 by 2.6%. With 32B base model, OpenSQL achieves 81.1% average EX, surpassing the best open-source model in Bird (ExCoT-DPO+Qwen-32B) by 1.6%. These results indicate that OpenSQL can robustly handle schema changes and new question types without retraining. This stems from OpenSQL's subtask-specific training, which allows models to focus on simpler objectives and makes them less sensitive to schema or question shifts than end-to-end training approaches.

Furthermore, we evaluated OpenSQL on different dialects using Bird-minidev (MySQL/PostgreSQL). Without dialect-specific tuning, OpenSQL-32B achieves 64.6% and 65.6% EX, outperforming leading baselines (*i.e.,* OmniSQL-32B and ExCoT+Qwen-32B). While these models suffer severe performance drops when shifting from SQLite to other dialects, OpenSQL shows superior robustness with much smaller decline. This stems from OpenSQL learning intrinsic SQL reasoning logic rather than fitting to SQLite-specific patterns.

7.3 Real-world Scenarios Study

Exp-3: How does OpenSQL perform in real-world scenarios? To evaluate the performance of OpenSQL in unseen real-world scenarios, we performed experiments on Kaggledbqa, Mimic-sql, Science, and Spider2-sqlite datasets. In this experiment, the test set was completely disjoint from the training data (Spider and Bird)

Table 2: Performance comparison (EX) on different datasets. "SC@N" means Self-Consistency with N samples. "–†" means the method's code is unavailable and did not report results. "–‡" means the result is unavailable due to training data contamination.

Method / Model	#Param	BIRD-dev	SPIDER-dev	SPIDER-test	KAGGLEDBQA	MIMIC	SCIENCE	SPIDER2-SQLITE
Prompting Methods with Closed-source LLMs								
DAIL-SQL [15] + GPT-4	>175B	54.8	83.6	86.6	46.0	36.1	48.2	10.4
MAC-SQL [69] + GPT-4	>175B	57.6	86.8	82.8	31.9	53.1	54.5	17.0
CHESS [67] + GPT-4	>175B	65.0	87.1	87.2	51.9	54.9	55.9	13.3
E-SQL [3] + GPT-4o	~200B	65.6	84.1	83.8	53.5	48.9	57.5	16.3
CodeS [35] + REDSQL [60] + GPT-4o	~200B	67.3	85.7	87.4	54.6	62.0	60.5	12.6
OpenSearch-SQL + GPT-4o	~200B	69.3	86.4	87.1	51.9	59.5	56.5	19.3
XiYan-SQL [16] + GPT-4o	~200B	73.3	–†	89.7	–†	–†	–†	–†
CHASE-SQL [50] + Gemini-1.5-Pro	>200B	74.5	–†	87.6	–†	–†	–†	–†
Text-to-SQL with Open-source Models								
ROUTE [55] + Qwen2.5-14B	14B	55.9	87.1	87.3	–†	–†	–†	–†
SFT CodeS + Schema Filter [35] + SC@24	15B	57.2	85.1	84.1	42.2	40.9	50.8	3.7
Qwen2.5-Coder [23] + SC@24	32B	64.8	83.4	84.1	31.9	61.3	59.2	17.8
XiYanSQL-Qwen2.5-Coder [16] + SC@24	32B	66.8	–‡	88.4	53.0	36.5	53.2	8.9
OmniSQL [33] + SC@24	32B	67.0	87.9	89.6	56.8	51.5	60.5	13.3
ExCoT-DPO [87] + Qwen2.5-Coder + SC@24	32B	68.7	86.4	86.7	60.5	47.7	60.9	13.3
SFT Qwen2.5-72B [82] + SC@24	72B	66.4	88.1	87.9	60.0	46.4	54.5	5.2
OpenSQL + Qwen2.5-Coder-7B	7B	67.2	**88.8**	86.9	60.5	62.8	56.6	8.9
OpenSQL + OmniSQL-7B	7B	69.5	87.6	86.8	61.1	61.4	60.5	**14.1**
OpenSQL + Llama3.1-8B	8B	65.1	86.8	86.6	60.0	64.1	58.9	5.9
OpenSQL + Qwen2.5-Coder-14B	14B	68.4	88.2	87.6	61.1	59.0	**62.5**	9.6
OpenSQL + Qwen2.5-Coder-32B	32B	**70.0**	88.6	**88.3**	**63.2**	**64.3**	61.5	**14.1**

to simulate real-world scenarios. Given the large scale of databases in this experiment, we set the SQL timeout to 600 seconds.

As shown in Table 2, OpenSQL consistently demonstrates competitive accuracy across out-of-domain real-world datasets. With Qwen-7B, OpenSQL achieves 60.5% EX on KAGGLEDBQA and 62.8% EX on MIMIC-SQL, surpassing all prompting-based baselines. With Qwen-32B, OpenSQL achieves better performance, including 61.5% EX on SCIENCE, matching the state-of-the-art. On the most challenging SPIDER2-SQLITE benchmark, OpenSQL combined with Qwen-32B and OmniSQL-7B both achieve 14.1% EX, outperforming OmniSQL-32B and ExCoT + Qwen2.5-Coder-32B at 13.3%. Notably, some open-source models trained on BIRD or SPIDER show degraded performance on the real-world benchmarks (*e.g.,* 5.2% EX of SFT Qwen2.5-72B on SPIDER2-SQLITE), whereas OpenSQL does not suffer from such degradation severely, highlighting its robustness under domain shift. Additionally, we find model size has a more pronounced impact on SPIDER2-SQLITE, with closed-source models overall outperforming open-source ones, indicating that complex schemas and queries require larger model capacity for effective reasoning. This experimental results indicate that OpenSQL delivers strong out-of-the-box accuracy without additional fine-tuning.

Exp-4: Can OpenSQL be deployed with limited models? How sensitive is OpenSQL to the data synthesis model? To evaluate OpenSQL's practicality under constrained resources, we examined the accuracy of OpenSQL with reduced model counts and weaker data synthesis models. Regarding deployment efficiency, we evaluated three consolidated settings: (1) Training a *Reasoner* model to perform SQL generation and SQL selection; (2) Further training a *Schema Linker* model to perform global-local schema linking; and (3) Training a single model for all tasks (excluding DPO to avoid catastrophic forgetting). As shown in Table 4, with a limited number of models in the multi-task training setting, OpenSQL maintains high accuracy. Using the *Reasoner* to perform SQL generation and selection results in virtually no EX loss. Across SPIDER and BIRD, the performance drop under multi-task training is less than 2%. This shows the adaptability of the OpenSQL pipeline with resource-constrained deployments. Regarding data robustness, replacing the data synthesis model with Gemma3-27B or Qwen3-32B yields consistent accuracy. This shows that while stronger LLMs offer a slight performance edge, our strategy of anchoring synthetic data to correct results ensures OpenSQL remains robust even with weaker teachers.

7.4 Data Efficiency Study

Exp-5: Can OpenSQL scale with more training data? And how well does OpenSQL perform under cold-start settings? To study the data efficiency capability of OpenSQL, we trained OpenSQL models on progressively larger subsets of the training data and evaluated the accuracy. Specifically, we created five subsets by randomly sampling 15%, 25%, 50%, 75%, and 100% of the data from the original training data, and trained OpenSQL with Qwen-2.5-Coder-7/32B and Llama3.1-8B as the base model. We evaluated the EX of OpenSQL trained on each split on the BIRD-dev and SPIDER-dev.

As shown in Figure 5, there is a positive correlation between the amount of training data and EX on both BIRD-dev and SPIDER-dev across different model architectures and parameter scales. This shows that OpenSQL scales effectively with additional training data. This is because more training data can strengthen each module's competence on the subtask and improve inter-module coordination. With 25% data, OpenSQL + Qwen-32B surpasses OmniSQL-32B, which uses 2.5 million (Question, SQL) pairs for training, demonstrating the data-efficiency of OpenSQL. Furthermore, we find that larger models perform better with limited data, but the accuracy gains become less prominent as training data increases.

Table 3: Comparison of EX accuracy (%) between OpenSQL and baseline methods on the Dr.spider dataset (upper) and Bird-minidev dataset of different dialects (lower).

Type of Perturbation / Dataset		Zero-NL2SQL +GPT-4	ExCoT+ Qwen-32B +SC@24	OmniSQL -32B +SC@24	OpenSQL Qwen -7B	OpenSQL Omni -7B	OpenSQL Llama -8B	OpenSQL Qwen -14B	OpenSQL Qwen -32B
DB	schema-synonym	71.4	80.3	**80.6**	78.2	75.8	76.3	78.5	79.8
	schema-abbreviation	75.9	84.9	86.0	83.8	84.8	84.1	**87.4**	86.2
	content-equivalence	**72.0**	68.6	63.9	68.3	71.5	64.1	69.9	69.4
NL	keyword-synonym	74.5	72.4	74.8	76.3	**77.8**	74.4	76.5	75.6
	keyword-carrier	89.0	87.0	89.2	91.5	90.2	**92.2**	91.0	91.2
	column-synonym	64.5	67.5	68.2	67.1	68.7	67.5	68.7	**70.5**
	column-carrier	73.2	81.9	83.4	83.2	82.7	81.9	**84.8**	83.1
	column-attribute	73.9	74.8	77.3	**80.7**	**80.7**	78.2	76.5	79.0
	column-value	74.3	82.9	80.3	80.3	77.6	78.6	**83.2**	82.6
	value-synonym	68.9	71.7	63.2	69.2	69.2	69.8	**72.7**	71.3
	multitype	67.6	70.5	70.7	70.6	71.9	70.2	**72.9**	72.2
	others	81.3	83.1	82.7	81.7	81.7	81.8	**83.6**	82.8
SQL	comparison	**79.2**	73.6	77.5	74.7	74.2	78.1	75.3	75.8
	sort-order	83.3	85.4	85.9	86.5	83.9	81.2	83.9	**89.1**
	nonDB-number	92.4	**94.7**	93.9	93.9	**94.7**	90.8	90.1	**94.7**
	DB-text	82.1	85.0	**86.2**	83.6	82.7	84.2	85.0	85.5
	DB-number	86.3	87.3	90.0	87.8	88.0	87.1	90.5	**90.7**
Dr.spider Average		77.2	79.5	79.6	79.8	79.8	78.9	80.6	**81.1**
Bird-minidev (MySQL)		–	47.8	57.0	60.6	60.0	59.2	61.6	64.6
Bird-minidev (PostgreSQL)		–	39.4	56.8	58.8	57.8	57.4	59.2	65.6

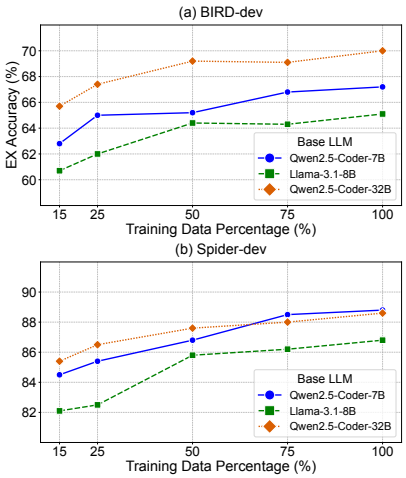

Figure 5: Accuracy of OpenSQL on Bird-dev (a) and Spider-dev (b) across different training data scales (15%-100%).

Table 4: OpenSQL-7B accuracy with resource-limited settings.

Setting	Bird-dev	Spider-dev
OpenSQL +Qwen-7B (Data synthesis w/ Deepseek-R1)	**67.2**	**88.8**
Global / Local Linker + Reasoner (3 models)	66.9	88.5
Schema Linker + Reasoner (2 models)	65.5	87.3
Unified Training (1 model)	65.6	87.7
Data synthesis w/ Gemma3-27B	67.0	87.7
Data synthesis w/ Qwen3-32B	66.8	88.0

Table 5: Overhead Analysis (On Bird-dev). Training Time are measured with 8 Nvidia H200 GPUs. Latency are measured with 4 Nvidia H200 GPUs due to vLLM compatibility.

Method / Stage	Train Time	Input Token	Output Token	Latency (s)	Avg. Cost	EX
CHASE-SQL + Gemini	UNK	~280K	~50K	58.7	$0.60	74.5
MAC-SQL + GPT-4	N/A	6.3K	0.6K	10.3	$0.22	57.6
E-SQL + GPT-4o	N/A	44K	1K	47.5	$0.12	65.6
CHESS + Llama3-70B	N/A	327K	24.8K	81.7	$0.31	61.5
SFT Qwen2.5-72B + SC@24	7.3h	4.5K	1.3K	3.3	$0.0046	66.4
SQL Generation	–	4.5K	1.3K	1.5	$0.0046	–
SQL Execution and Vote	–	–	–	1.8	$0	–
OpenSQL+Qwen2.5-Coder-7B	**3.6h**	**9.8+30K**	**3.0K**	**9.6**	**$0.0035**	**67.2**
Global Schema Linking	1.4h	4.5K	0.05K	0.25	$0.0006	–
Local Schema Linking	0.7h	30K	0.05K	0.32	$0.0013	–
SQL Generation	1.0h	1.8K	1.1K	0.36	$0.0006	–
SQL Execution	–	–	–	5.12	–	–
Tournament Selection	0.5h	3.5K	1.8K	2.84	$0.0010	–
OpenSQL + Qwen2.5-Coder-14B	**6.4h**	**9.6K+30K**	**2.9K**	**10.6**	**$0.0064**	**68.4**
Global Schema Linking	3.0h	4.5K	0.05K	0.37	$0.0013	–
Local Schema Linking	0.7h	30K	0.05K	0.33	$0.0013	–
SQL Generation	1.8h	1.8K	1.1K	0.49	$0.0014	–
SQL Execution	–	–	–	5.32	–	–
Tournament Selection	0.9h	3.3K	1.7K	4.10	$0.0024	–

Exp-6: How are the training time, inference cost, and latency of OpenSQL? We further analyzed the cost of OpenSQL to validate its cost-effectiveness. We computed the average number of input and output tokens per question on Bird-dev and converted them into costs with the official API price. Although OpenSQL can be deployed locally, we included the cost converted from the Qwen API price as references. We compared OpenSQL with prompting-based methods built on different LLMs, including CHASE-SQL (Gemini), MAC-SQL (GPT-4), E-SQL (GPT-4o), and CHESS (Llama3-70B). We also directly fine-tuned a larger LLM, Qwen2.5-72B, using the Spider-train and Bird-train datasets, under the same experimental settings.

As shown in Table 5, compared with CHASE-SQL, which relies on prompting Gemini-1.5-Pro, OpenSQL reduces both input and output tokens, leading to a cost reduction by two orders of magnitude. This is because OpenSQL does not require few-shot examples in the LLM context. Compared to E-SQL, MAC-SQL, OpenSQL achieves higher EX accuracy with low inference cost. We also evaluate OpenSQL's training time and latency. Notably, we observe that OpenSQL exhibits markedly lower latency than advanced prompting-based closed-source approaches. The tournament selection introduces additional inference latency, but the increase is not significant. This is because the majority of SQL candidates yield identical execution results and do not incur pairwise comparisons. As a result, the inference latency is dominated by the execution of SQL candidates, which is adopted by most Text-to-SQL methods. Finally, we observe that OpenSQL generates explicit reasoning structures (*e.g.,* CTE), which may result in more complex SQL queries that incur longer execution time on benchmark workloads. Future work may take the execution efficiency of synthesized SQL into consideration.

7.5 Ablation Study

Exp-7: What is the impact of each module in OpenSQL? We conducted an ablation study with Qwen-7B and Qwen-32B to evaluate the contribution of each component. Results are shown in Table 6.

Progressive Ablation. The upper section of Table 6 reports a progressive ablation where we iteratively remove modules from the full pipeline. First, removing the global-local schema linking module leads to an accuracy drop. This confirms that precise schema pruning is essential for reducing input noise. Second, further removing the Diversified SQL Generator (using single-path generation) reduces accuracy, showing the effectiveness of diverse reasoning paths. Third, removing the Stepwise Selection module causes the most significant decline, demonstrating that accurately selecting the answer from multiple tries is critical for Text-to-SQL.

Table 6: Ablation study of OpenSQL. The primary metric is EX. At the right of EX is a recall for the correct SQL.

Configuration	Bird-dev		Spider-dev	
	7B	32B	7B	32B
Full Pipeline	**67.2**(77.1)	**70.0**(78.8)	**88.8**(91.6)	**88.6**(92.2)
– Global-Local Schema Linking	66.0	68.7	86.8	88.2
– Diversified Generation	64.7	67.5	86.3	88.0
– Stepwise Selection	59.6	64.1	84.3	87.1
Ablation on Schema Linking (EX)				
only Global Linker (w/ DPO)	66.6	69.2	87.8	88.2
only Global Linker (w/o DPO)	65.5	68.7	87.0	87.9
Ablation on Diversified Generation (EX / Recall)				
w/o Reasoning Path SUBQ	65.9(74.9)	69.4(77.1)	88.2(90.9)	88.4(91.3)
w/o Reasoning Path CTE	66.5(75.0)	69.2(77.5)	88.2(91.5)	88.3(91.9)
Ablation on Selection Strategy (EX)				
w/ Self-Consistency	65.3	68.6	86.8	87.9
w/ Direct Pairwise Comparison	65.5	69.0	87.9	88.3

Component Analysis. The lower section of Table 6 validates key design choices by replacing one component with a simpler alternative while keeping the rest unchanged. (i) For schema linking, relying solely on the global linker decreases accuracy, and further removing DPO causes a sharper drop to 65.5%. This confirms that the local linker is essential for fine-grained column alignment, while DPO enhances the global linker's ability to retain necessary schema items. (ii) Regarding generation, excluding the SUBQ or CTE path reduces the recall of correct SQL and subsequently the final EX, validating that diverse syntactic structures complement each other to cover complex reasoning logic. (iii) Finally, for selection, our stepwise strategy outperforms both self-consistency and direct pairwise comparison, showing that structured clause-level reasoning is critical for distinguishing correct queries from subtle semantic errors.

Discussion on Model Scale and Difficulty. The ablation results confirm that our proposed modules are consistently effective across varying model scales, while the magnitude of improvement varies depending on the interplay between model capacity and task difficulty. Specifically, the relative gains on the 32B model are slightly narrower compared to the 7B counterpart, reflecting the stronger inherent reasoning capabilities of the larger backbone. Additionally, we observe a saturation effect on the easier Spider-dev benchmark, where the marginal gains from optimizations are relatively limited.

8 DISCUSSION AND LIMITATIONS

Justification for Small Fine-Tuned Models. Small fine-tuned models offer advantages in terms of lower long-term costs, higher accuracy, and customization. While serving large LLMs incurs substantial resource overhead, small models allow for cost-effective long-term deployment due to lower VRAM requirements and less token consumption. Through task-specific training, small models can surpass larger general models in accuracy. Furthermore, the minimal training and deployment overhead facilitate efficient customization within privacy-critical environments.

Reliability and Applicability of the Augmented Data. We ensure the reliability of augmented sub-task supervision by using the execution correctness of the final SQL as validation. This rigorous alignment guarantees the validity of the synthesized logic, thereby minimizing spurious correlations or artifacts. Furthermore, our framework requires only standard end-to-end (Question, SQL) pairs. It can automatically synthesize the intermediate supervision from these pairs, allowing OpenSQL to seamlessly adapt to diverse datasets and SQL dialects with minimal manual intervention. Although a risk persists where flawed intermediate reasoning might yield the correct result, our experiments with varying synthesis models demonstrate negligible performance decline.

Serving Choices. OpenSQL supports flexible deployment, allowing for either modular serving or a single unified model (as in Table 4). The unified option can ensure a controllable memory footprint and limit the operational overhead to that of a single small model.

Trade-off on Developer Effort. OpenSQL requires training multiple specialized modules, which involves higher initial engineering effort compared to end-to-end approaches. However, this modular specificity is designed to maximize the potential of small open-source models. We observed that Text-to-SQL sub-tasks possess distinct characteristics best addressed by tailored training paradigms. Accordingly, we implement specialized training strategies for each module to maximize the overall accuracy. Consequently, the increased training complexity is justified by the gains in inference accuracy and generalization ability. Future iterations could leverage reinforcement learning to unify the training pipeline. Regarding maintenance, adapting to new domains requires only collecting new (Question, SQL) pairs. These pairs are processed by an automated data augmentation pipeline to facilitate *incremental fine-tuning*. Consequently, developers can efficiently maintain the system by either fine-tuning a single unified model or selectively updating specific modules without retraining the entire system.

Limitation on Schema Scalability. The global schema linker takes the full database schema as input to leverage the model's comprehension capability. This may face limitations in enterprise scenarios with extremely wide tables or thousands of columns. In such cases, the full schema may exceed the context window of small LLMs. To mitigate this, future iterations could adopt a schema partitioning strategy to process complex schemas multiple times.

9 CONCLUSION

We presented OpenSQL, a data-efficient framework that enhances open-source LLMs for Text-to-SQL. OpenSQL synthesizes intermediate supervision from limited annotations and trains the model to master intermediate Text-to-SQL sub-tasks. Experiments show that OpenSQL outperforms open-source models with less training data and competes with advanced closed-source LLMs.

ACKNOWLEDGMENTS

This work was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (62525202, 62232009, 62402409), Shenzhen Project (CJGJZD20230724093403007), China Railway Science Research Institute Group Co., Ltd, Zhongguancun Lab, Huawei, Tsinghua-Tencent Joint Laboratory and Beijing National Research Center for Information Science and Technology (BNRist). Guoliang Li is the corresponding author.

REFERENCES

[1] [n.d.]. TRL - Transformer Reinforcement Learning. https://github.com/huggingface/trl. Accessed: 2025-11-26.

[2] Mohammad Ali Alomrani, Yingxue Zhang, Derek Li, Qianyi Sun, Soumyasundar Pal, Zhanguang Zhang, Yaochen Hu, Rohan Deepak Ajwani, Antonios Valkanas, Raika Karimi, Peng Cheng, Yunzhou Wang, Pengyi Liao, Hanrui Huang, Bin Wang, Jianye Hao, and Mark Coates. 2025. Reasoning on a Budget: A Survey of Adaptive and Controllable Test-Time Compute in LLMs. arXiv:2507.02076 [cs.AI] https://arxiv.org/abs/2507.02076

[3] Hasan Alp Caferoğlu and Özgür Ulusoy. 2024. E-SQL: Direct Schema Linking via Question Enrichment in Text-to-SQL.

[4] 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 (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 2541–2555.

[5] Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation.

[6] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1335–1349. https://doi.org/10.1145/3318464.3389742

[7] Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. 2024. SQLFixAgent: Towards Semantic-Accurate Text-to-SQL Parsing via Consistency-Enhanced Multi-Agent Collaboration.

[8] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, et al. 2023. Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness.

[9] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.

[10] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. TURL: Table Understanding through Representation Learning. *SIGMOD Rec.* 51, 1 (June 2022), 33–40. https://doi.org/10.1145/3542700.3542709

[11] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library.

[12] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, and et al. 2024. The Llama 3 Herd of Models.

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

[14] 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 (Feb. 2023), 1534–1547. https://doi.org/10.14778/3583140.3583165

[15] 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.

[16] 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. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL.

[17] Michael Glass, Mustafa Eyceoz, Dharmashankar Subramanian, Gaetano Rossiello, Long Vu, and Alfio Gliozzo. 2025. Extractive Schema Linking for Text-to-SQL. arXiv:2501.17174 [cs.DB] https://arxiv.org/abs/2501.17174

[18] Gemini Team Google. 2023. Gemini: A Family of Highly Capable Multimodal Models.

[19] Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. 2025. MSc-SQL: Multi-Sample Critiquing Small Language Models For Text-To-SQL Translation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Luis Chiruzzo, Alan Ritter, and Lu Wang (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 2145–2160. https://doi.org/10.18653/v1/2025.naacl-long.107

[20] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2024. Think before you speak: Training Language Models With Pause Tokens. In *The Twelfth International Conference on Learning Representations*.

[21] 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, Article 147 (June 2023), 28 pages. https://doi.org/10.1145/3589292

[22] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. 2025. Training Large Language Models to Reason in a Continuous Latent Space. In *Second Conference on Language Modeling*. https://openreview.net/forum?id=Itxz7S4Ip3

[23] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, and et.al. 2024. Qwen2.5-Coder Technical Report.

[24] Eunki Kim, Sangryul Kim, and James Thorne. 2025. Learning to Insert [PAUSE] Tokens for Better Reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 23760–23777. https://doi.org/10.18653/v1/2025.findings-acl.1217

[25] 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* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. https://doi.org/10.1145/3600006.3613165

[26] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 2261–2273.

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

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

[29] Zhenyu Lei, Zhen Tan, Song Wang, Yaochen Zhu, Zihan Chen, Yushun Dong, and Jundong Li. 2025. Learning from Diverse Reasoning Paths with Routing and Collaboration. arXiv:2508.16861 [cs.CL] https://arxiv.org/abs/2508.16861

[30] Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. 2025. DeepEye-SQL: A Software-Engineering-Inspired Text-to-SQL Framework. *CoRR* abs/2510.17586 (2025).

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

[32] 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. In *ICML*. OpenReview.net.

[33] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale. arXiv:2503.02240 [cs.CL] https://arxiv.org/abs/2503.02240

[34] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: decoupling schema linking and skeleton parsing for text-to-SQL. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, Article 1466, 9 pages.

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

[36] 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 *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, Article 1467, 9 pages.

[37] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2024. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 1835, 28 pages.

[38] Xiuwen Li, Qifeng Cai, Yang Shu, Chenjuan Guo, and Bin Yang. 2025. AID-SQL: Adaptive In-Context Learning of Text-to-SQL with Difficulty-Aware Instruction and Retrieval-Augmented Generation . In *2025 IEEE 41st International Conference*

on Data Engineering (ICDE). IEEE Computer Society, Los Alamitos, CA, USA, 3945–3957. https://doi.org/10.1109/ICDE65448.2025.00294

[39] Yuliang Li, Jinfeng Li, Yoshi Suhara, AnHai Doan, and Wang-Chiew Tan. 2023. Effective entity matching with transformers. *The VLDB Journal* 32, 6 (Jan. 2023), 1215–1235. https://doi.org/10.1007/s00778-023-00779-z

[40] Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making Language Models Better Reasoners with Step-Aware Verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 5315–5333. https://doi.org/10.18653/v1/2023.acl-long.291

[41] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 4870–4888. https://doi.org/10.18653/v1/2020.findings-emnlp.438

[42] Hanbing Liu, Haoyang Li, Xiaokang Zhang, Ruotong Chen, Haiyong Xu, Tian Tian, Qi Qi, and Jing Zhang. 2025. Uncovering the Impact of Chain-of-Thought Reasoning for Direct Preference Optimization: Lessons from Text-to-SQL. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 21223–21261. https://doi.org/10.18653/v1/2025.acl-long.1031

[43] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? arXiv:2408.05109 [cs.DB] https://arxiv.org/abs/2408.05109

[44] Ruilin Luo, Liyuan Wang, Binghuai Lin, Zicheng Lin, and Yujiu Yang. 2024. PTD-SQL: Partitioning and Targeted Drilling with LLMs in Text-to-SQL. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Miami, Florida, USA, 3767–3799.

[45] Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning. arXiv:2504.08600 [cs.DB] https://arxiv.org/abs/2504.08600

[46] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 19–34. https://doi.org/10.1145/3183713.3196926

[47] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling.

[48] OpenAI. 2023. GPT-4 Technical Report.

[49] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

[50] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL.

[51] Mohammadreza Pourreza and Davood Rafiei. 2024. DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 1577, 10 pages.

[52] Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, Miami, Florida, USA, 8212–8220.

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

[54] 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*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 3215–3229.

[55] Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2025. ROUTE: Robust Multitask Tuning and Collaboration for Text-to-SQL. In *The Thirteenth International Conference on Learning Representations*.

[56] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In *Findings of the Association for Computational Linguistics ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 5456–5471.

[57] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Thirty-seventh Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 2338, 14 pages.

[58] Jonathan Raiman and Olivier Raiman. 2018. DeepType: multilingual entity linking by neural type system evolution *(AAAI'18/IAAI'18/EAAI'18)*. AAAI Press, Article 663, 8 pages.

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

[60] Tonghui Ren, Chen Ke, Yuankai Fan, Yinan Jing, Zhenying He, Kai Zhang, and X. Sean Wang. 2025. The Power of Constraints in Natural Language to SQL Translation. *Proc. VLDB Endow.* 18, 7 (2025).

[61] Gaetano Rossiello, Nhan H Pham, Michael Glass, Junkyu Lee, and Dharmashankar Subramanian. 2025. Rationalization Models for Text-to-SQL. In *Workshop on Reasoning and Planning for Large Language Models*. https://openreview.net/forum?id=QgUVtHcHzI

[62] Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 311–324. https://doi.org/10.18653/v1/2021.naacl-main.29

[63] Ritesh Sarkhel and Arnab Nandi. 2024. Cross-Modal Entity Matching for Visually Rich Documents. arXiv:2303.00720 [cs.LG] https://arxiv.org/abs/2303.00720

[64] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. arXiv:2408.03314 [cs.LG] https://arxiv.org/abs/2408.03314

[65] Michael Stonebraker and Andrew Pavlo. 2024. What Goes Around Comes Around... And Around... *SIGMOD Record.* 53, 2 (2024), 21–37.

[66] Ji Sun, Guoliang Li, Peiyao Zhou, Yihui Ma, Jingzhe Xu, and Yuan Li. 2025. AgenticData: An Agentic Data Analytics System for Heterogeneous Data. arXiv:2508.05002 [cs.DB] https://arxiv.org/abs/2508.05002

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

[68] Fengwei Teng, Zhaoyang Yu, Quan Shi, Jiayi Zhang, Chenglin Wu, and Yuyu Luo. 2025. Atom of Thoughts for Markov LLM Test-Time Scaling. *CoRR* abs/2502.12018 (2025).

[69] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2023. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL.

[70] 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*. Association for Computational Linguistics, Online, 7567–7578.

[71] Jiayi Wang and Guoliang Li. 2025. AOP: Automated and Interactive LLM Pipeline Orchestration for Answering Complex Queries. In *CIDR*. www.cidrdb.org. https://vldb.org/cidrdb/2025/aop-automated-and-interactive-llm-pipeline-orchestration-for-answering-complex-queries.html

[72] Jiayi Wang, Guoliang Li, and Jianhua Feng. 2025. iDataLake: An LLM-Powered Analytics System on Data Lakes. *IEEE Data Eng. Bull.* 49, 1 (2025), 57–69. http://sites.computer.org/debull/A25mar/p57.pdf

[73] Jiayi Wang, Yuan Li, Jianming Wu, Shihui Xu, and Guoliang Li. 2025. Unify: A System For Unstructured Data Analytics. *Proc. VLDB Endow.* 18, 12 (2025), 5287–5290. https://doi.org/10.14778/3750601.3750653

[74] Ping Wang, Tian Shi, and Chandan K. Reddy. 2020. Text-to-SQL Generation for Question Answering on Electronic Medical Records. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 350–361. https://doi.org/10.1145/3366423.3380120

[75] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*.

[76] 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 *Proceedings of the 36th International Conference on Neural Information Processing Systems*.

[77] Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Scalable Zero-shot Entity Linking with Dense Entity Retrieval. In

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 6397–6407. https://doi.org/10.18653/v1/2020.emnlp-main.519

[78] Jinyu Xiang, Jiayi Zhang, Zhaoyang Yu, Fengwei Teng, Jinhao Tu, Xinbing Liang, Sirui Hong, Chenglin Wu, and Yuyu Luo. 2025. Self-Supervised Prompt Optimization. *CoRR* abs/2502.06855 (2025).

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

[80] Yuanzhen Xie, Xinzhou Jin, Tao Xie, Mingxiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for Enhancing Attention: Improving LLM-based Text-to-SQL through Workflow Paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 10796–10816.

[81] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. 2020. LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 6442–6454. https://doi.org/10.18653/v1/2020.emnlp-main.523

[82] Qwen: An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[83] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 517, 14 pages.

[84] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 3911–3921.

[85] Zheng Yuan, Hao Chen, Zijin Hong, Qinggang Zhang, Feiran Huang, Qing Li, and Xiao Huang. 2025. Knapsack Optimization-based Schema Linking for LLM-based Text-to-SQL Generation. arXiv:2502.12911 [cs.CL] https://arxiv.org/abs/2502.12911

[86] Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah Goodman. 2024. Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking. In *First Conference on Language Modeling*.

[87] Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. 2025. Optimizing Reasoning for Text-to-SQL with Execution Feedback. In *Findings of the Association for Computational Linguistics: ACL 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 19206–19218. https://doi.org/10.18653/v1/2025.findings-acl.982

[88] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, Singapore, 3501–3532.

[89] Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Association for Computational Linguistics, Miami, Florida, US, 1393–1412.

[90] 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 (Dec. 2023), 685–698. https://doi.org/10.14778/3636218.3636225

[91] 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 *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2020, 29 pages.

[92] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. DB-GPT: Large Language Model Meets Database. *Data Sci. Eng.* 9, 1 (2024), 102–111. https://doi.org/10.1007/S41019-023-00235-6

[93] Yizhang Zhu, Shiyin Du, Boyan Li, Yuyu Luo, and Nan Tang. 2024. Are large language models good statisticians? *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 2005, 35 pages.

[94] Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, Wei Zhou, Xinyu Liu, Zhangyang Peng, Tianqi Luo, Yu Li, et al. 2025. A Survey of Data Agents: Emerging Paradigm or Overstated Hype? *arXiv preprint arXiv:2510.23587* (2025).

[95] Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, Wei Zhou, Xinyu Liu, Zhangyang Peng, Tianqi Luo, Yu Li, Chengliang Chai, Chong Chen, Shimin Di, Ju Fan, Ji Sun, Nan Tang, Fugee Tsung, Jiannan Wang, Chenglin Wu, Yanwei Xu, Shaolei Zhang, Yong Zhang, Xuanhe Zhou, Guoliang Li, and Yuyu Luo. 2025. A Survey of Data Agents: Emerging Paradigm or Overstated Hype? *CoRR* abs/2510.23587 (2025). https://doi.org/10.48550/ARXIV.2510.23587 arXiv:2510.23587