Data-Centric Text-to-SQL with Large Language Models

Zezhou Huang
Columbia University
zh2408@columbia.edu

Shuo Zhang
Columbia University
sz3177@columbia.edu

Kechen Liu
Columbia University
kl3469@columbia.edu

Eugene Wu
DSI, Columbia University
ewu@cs.columbia.edu

Abstract

Text-to-SQL is crucial for enabling non-technical users to access data, and large language models have significantly improved performance. However, recent frameworks are largely Query-Centric, focusing on improving models' ability to translate natural language into SQL queries. Despite these advancements, real-world challenges—especially messy and large datasets—remain a major bottleneck. Our case studies reveal that 11-37% of the ground truth answers in the BIRD benchmark are incorrect due to data quality issues (duplication, disguised missing values, data types and inconsistent values). To address this, we propose a Data-Centric Text-to-SQL framework that preprocesses and cleans data offline, builds a relationship graph between tables, and incorporates business logic. This allows LLM agents to efficiently retrieve relevant tables and details during query time, significantly improving accuracy. Our experiments show that this approach outperforms human-provided ground truth answers on the BIRD benchmarks by up to 33.89%.

1 Introduction

Text-to-SQL is a widely used task that enables domain experts who are not familiar with databases or SQL to access data easily. Recent Text-to-SQL frameworks [10, 16, 3, 24], based on off-the-shelf large language models like GPT-4 [1] and Claude 3.5, with trillions of parameters, have demonstrated state-of-the-art results on public benchmarks such as BIRD [19], Spider [28], and KaggleDBQA [14].

Current Text-to-SQL frameworks are **Query-Centric**, focusing mainly on enhancing models' ability to translate natural language into SQL queries. For example, previous works have improved query results by using better fine-tuned models [17, 18, 8], implementing self-debugging frameworks [2], or decomposing tasks for individual SQL components [24, 27, 15]. However, in real-world applications, a major challenge—beyond just constructing queries—is dealing with messy and large datasets, which complicates the query construction process. The models need to not only understand the SQL language and map the concepts but also search for the correct tables and perform cleaning on the fly:

- **Messy:** Text-to-SQL solutions are mostly needed for raw data sources, such as those in an enterprise data lake [22], where data has not yet been transformed into reporting models needed for answering queries. However, these raw data sources often have issues like duplication, missing values, or inconsistencies [5], which can negatively impact the performance of Text-to-SQL.
- **Large:** Enterprise data often consists of a large number of data sources [21, 9]. Even within a single data source, it's common to find hundreds or even thousands of tables with complex join and union relationships, e.g., for ERP and CRM systems [7]. To link the query to the table schema, previous works have represented the schema in different structures [8, 25]. Recent works schema

Table Representation Learning Workshop at NeurIPS 2024.

linking is less needed for latest LLMs [20]. However, all these works study relatively small databases with < 20 tables, while enterprise data lakes have tables numbering in the thousands.

To illustrate how data can be a bottleneck that complicates text-to-SQL tasks, consider the example tables and queries from the BIRD benchmark [19], as shown in Figure 1. Duplication leads to incorrect aggregation queries, such as AVG. Disguised missing values and inconsistent value representations make filter conditions challenging by requiring additional predicates. Additionally, column types may need to be cast before performing aggregation. These issues are not mere edge cases. Our case study over 4 databases in BIRD reveals that, surprisingly, **11-37% of the ground truth answers are incorrect**, because the provided SQL queries don't handle these issues. In addition to obviously incorrect answers, there are other data issues like missing values and referential integrity violations that are concerning, though we cannot confirm whether they affect the correctness of the answers.

We propose a **Data-Centric** Text-to-SQL framework that leverages large language models to pre-process data. This is **an offline process independent of online queries that prepares the RAG structure and cleans the table**. It complements the Model-Centric framework online by helping with schema linking over large tables and eliminating the need for cleaning on the fly. Offline, the Data-Centric framework (1) profiles and cleans data to address quality issues, (2) builds a relationship graph to identify join and union paths between tables, and (3) summarizes these tables from different data sources, incorporating business logic. Throughout the process, we utilize existing enterprise text documents, data pipeline code (using embedding-based RAG) and human feedback. Online, when a natural language query is made, the framework provides APIs for query-centric LLM agents to follow an "overview to zoom-in" approach: (1) first use RAG APIs to find relevant data sources and tables, (2) then call functions to retrieve table and column details (e.g., descriptions, value representations, data types, etc.) to finalize query construction. We present our early experiment results on the Bird benchmarks, which show that, such data-centric framework, combined with a naive agent for online query, outperformed human-provided answers by up to 33.89%.

2 Data-Centric framework

In this section, we describe the details of the Data-Centric LLM Framework. We start with the offline data preparation, then the APIs it exposes for online Query-Centric Frameworks.

- **Data cleaning**: We begin by identifying and cleaning the data. Since data cleaning is too complex for current LLMs, we break it down into specific tasks such as handling duplicates, disguised missing values, data types, and value representations across tables and columns [11, 29, 12]. Our previous experiments have shown better performance compared to existing data cleaning tools.
- **Integration**: We construct a relationship graph by identifying joinable and unionable paths [26]. For union, we focus on cases where there is a 1-1 correspondence between columns for table partitions. We embed the schema and sample rows, then use LLMs to determine the 1-1 correspondence for closest tables. Note that columns can be only partially corresponded or require transformations, which we leave for future work. For joins, we break down the task: First, we identify primary and foreign keys for all tables. Then, for each foreign key, we use LLMs to find the corresponding primary keys, considering factors such as table features, column name descriptions, and column values. Additionally, we treat time and spatial columns as special cases. For example, using time as a join key can make the graph too dense to navigate since time is a common attribute.
- **Modeling**: The relationship graph can be too large to explore fully. To address this, we follow the Graph RAG preparation process [4] by clustering nodes into communities using the Leiden algorithm. We then generate natural language summaries for each community, incorporating relevant business logic from existing documents. For example, in a relationship graph of Salesforce databases [6] with about 500 tables, the summarized communities include "User Management and Authentication," "Customer Relationship Management," and "Financial and Transactional."

Online, we offer APIs for a Query-Centric Framework that follows an "overview-to-detail" approach. Starting with a natural language query, we first identify the most relevant communities and tables (through RAG), then zoom in on table specifics to complete the SQL construction.

```sql
-- Wrong (includes duplicates)
SELECT AVG(Sentiment) FROM user_reviews
-- Correct (deduplicates first)
SELECT AVG(Sentiment) FROM
(SELECT DISTINCT * FROM user_reviews)
```

user_reviews

Review	App	Sentiment
I like diet	Diet	0.9
I like diet	Diet	0.9

(a) Duplication causes a direct average aggregation query without deduplication to be wrong.

```sql
-- Wrong (misses 'nan' values)
SELECT AVG(Sentiment) FROM user_reviews
WHERE Review IS NOT NULL
-- Correct (handles both NULL and 'nan')
SELECT AVG(Sentiment) FROM user_reviews
WHERE Review IS NOT NULL AND Review != 'nan'
```

user_reviews

Review	App	Sentiment
NULL	Diet	0.7
nan	Diet	0.8

(b) Disguised missing values cause the IS NULL filter without considering other variants to be wrong.

```sql
-- Wrong (doesn't cast TEXT to INT)
SELECT AVG(PPA) FROM Scoring
-- Correct (casts TEXT to INT)
SELECT AVG(CAST(PPA AS INT)) FROM Scoring
```

Scoring

playerID	PPA (TEXT)
1001	"15"
1002	"18"

(c) Incorrect data types cause a direct average aggregation query without proper type casting to be wrong.

```sql
-- Wrong (doesn't account for variations)
SELECT dba_name FROM establishment
WHERE city = 'Chicago'
-- Correct (accounts for variations)
SELECT dba_name FROM establishment
WHERE city IN ('Chicago', 'CHICAGO')
```

establishment

dba_name	city
Mcdonald	Chicago
KFC	CHICAGO

(d) Inconsistent data representation causes a selection query that doesn't account for variations to be wrong.

Figure 1: Data quality issues in BIRD benchmark leading to incorrect ground truth answers

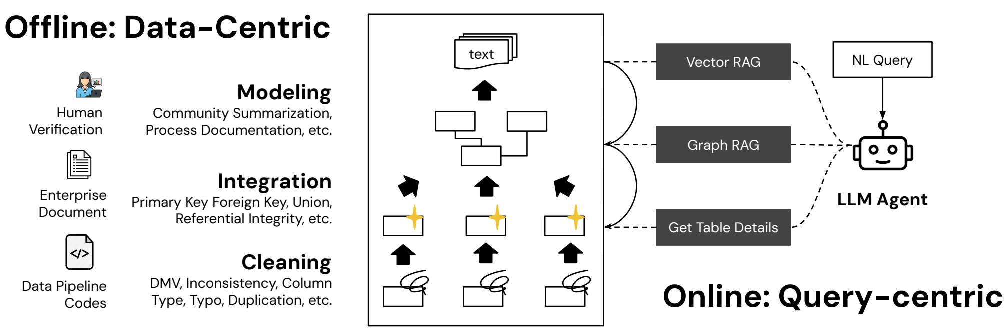

Figure 2: Data-Centric Text-to-SQL framework: Offline, prepares data through cleaning, integration, and modeling. Online, exposes APIs for Query-Centric framework to navigate data and write SQLs.

- **Vector RAG**: For each community (across data sources), we embed its natural language summary, allowing the agent to find the closest match based on the query within the embedding space.
- **Graph RAG**: Once relevant communities are detected, we provide Graph RAG APIs that (1) output all tables and their descriptions, helping the LLM to decide which tables are relevant for schema linking [17]; (2) for any table, return all K-hop neighbors within the relationship graph (including from other communities); and (3) given a set of tables, find the minimum spanning tree to identify potential join/union paths between source tables. For each join path, we also provide APIs to check referential integrity violations and join multiplicity to avoid fanout issues.
- **Table Details**: We provide details of: (1) table and column descriptions, and sample data [13]; (2) column representations: For numerical data, we provide the $0^{th}, 25^{th}, 50^{th}, 75^{th}, 100^{th}$

percentiles. For categorical data, we list the categories. For free text, we return regex patterns if available, or sample values otherwise; (3) data quality issues like missing values.

3 Experiments

3.1 Setup

To evaluate our data-centric framework, we use the BIRD dataset [19], which contains real-world data quality issues. The limit of BIRD is that the data sources are small, with <20 tables. Therefore, our experiment focuses on how data cleaning affects accuracy. For the query agent, we use a naive implementation that first identifies the community (Vector RAG), then finds the related tables and their join paths (Graph RAG), explores all relevant table details, and finally generates the SQL.

For the evaluation, we compare the results to ground-truth SQL queries (run on cleaned data). If they differ, we manually review if the queries are semantically equivalent in 3 cases: handling ties, the order of columns in the SELECT clause, and the reasonable inclusion of extra columns [23, 15].

To demonstrate our framework's effectiveness on large enterprise datasets, we present modeling results for Fivetran data sources at `https://cocoon-data-transformation.github.io/page/model`. The largest data source (Salesforce) contains ~ 500 tables. Since no text-to-SQL benchmark exists for these data, we leave the accuracy assessment for future work.

3.2 Results

Table 1: SQL correctness. By cleaning data, Data-Centric framework outperforms ground truth.

System	Hockey	Food Inspection 2	App Store	Human Resources
Ground Truth	77.30%	88.49%	63.50%	62.72%
Data-Centric	**82.13%**	**89.93%**	**76.19%**	**96.61%**

Table 2: Distribution of Data Quality Issues causing Incorrect Ground Truth Answers

Dataset	DMV	Inconsistency	Column Type	Duplication
Hockey	2.90%	5.31%	14.49%	–
Food Inspection 2	–	9.35%	2.16%	–
App Store	4.76%	3.17%	3.17%	25.40%
Human Resources	1.69%	–	35.59%	–

Across the 4 databases we analyzed, the accuracy of the provided ground truth queries and those generated by the Data-Centric Framework are summarized in Table 1. The types of data quality issues impacting the accuracy of the ground truth are shown in Table 2. Key findings include:

- **11-37% of the provided ground truth answers are incorrect due to data quality issues.** Data quality problems are widespread in BIRD, greatly affecting SQL accuracy. The most common issue across all four databases is the improper column type. Additionally, 3 of the 4 databases have disguised missing values and inconsistent data representation (including typos). In the App Store database, duplication affects 25.4% of the ground truth queries. Finally, we find that all 4 databases contain missing values in columns, while Hockey and Food Inspection 2 have referential integrity violations. These may include incomplete records that the data cleaning process cannot resolve, and their impact on query results remains unknown.
- **The Data-Centric Framework's queries outperform the ground truth queries by up to 33.89%.** Our findings show that preparing the data (cleaning, integration and modeling), even when done once and independently from the queries, can significantly boost the performance of online SQL queries, surpassing the provided ground truth. The level of improvement varies across different databases, with those having more severe data quality issues showing the greatest gains. It's important to note that this experiment used a basic Query-Centric Framework without advanced components for individual SQL optimization or self-debugging, which previous studies

have demonstrated to improve performance significantly [2, 24, 27, 15]. We anticipate even better results once we incorporate these frameworks, which will be explored in future work.

4 Conclusion

This work proposes a Data-Centric Text-to-SQL framework that complements existing Query-Centric approaches by addressing data quality and complexity issues in large and messy datasets. We find that preprocessing data, building relationship graphs, and incorporating business logic significantly improves query accuracy, outperforming human-provided answers on the BIRD benchmark by up to 33.89%. Future work will focus on integrating this data-centric approach with existing query-centric frameworks and evaluating the combined system on larger enterprise-level Text-to-SQL benchmarks.

Acknowledgements

This research received funding from multiple sources, including National Science Foundation grants (NSF #1845638, #1740305, #2008295, #2106197, #2103794) as well as corporate support from Amazon, Google, Adobe, and CAIT. The views and conclusions presented here are those of the authors and should not be interpreted as representing the official positions of the funding organizations.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[3] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*, 2023.

[4] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[5] Lisa Ehrlinger and Wolfram Wöß. A survey of data quality measurement and monitoring tools. *Frontiers in big data*, 5:850611, 2022.

[6] Fivetran. Salesforce. `https://fivetran.com/docs/connectors/applications/salesforce`, n.d. Fivetran Documentation.

[7] Helena Forslund. Erp systems' capabilities for supply chain performance management. *Industrial Management & Data Systems*, 110(3):351–367, 2010.

[8] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*, 2023.

[9] Corinna Giebler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. Leveraging the data lake: current state and challenges. In *Big Data Analytics and Knowledge Discovery: 21st International Conference, DaWaK 2019, Linz, Austria, August 26–29, 2019, Proceedings 21*, pages 179–188. Springer, 2019.

[10] Gunther Hagleitner. Gpt-4's sql mastery. `https://medium.com/querymind/gpt-4s-sql-mastery-2cd1f3dea543`, April 2023.

[11] Zezhou Huang and Eugene Wu. Cocoon: Semantic table profiling using large language models. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*, pages 1–7, 2024.

[12] Zezhou Huang and Eugene Wu. Relationalizing tables with large language models: The promise and challenges. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2024.

[13] Zezhou Huang, Pavan Kalyan Damalapati, and Eugene Wu. Data ambiguity strikes back: How documentation improves gpt's text-to-sql. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

[14] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. Kaggledbqa: Realistic evaluation of text-to-sql parsers. *arXiv preprint arXiv:2106.11455*, 2021.

[15] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv preprint arXiv:2405.07467*, 2024.

[16] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. The dawn of natural language to sql: Are we fully ready? *arXiv preprint arXiv:2406.01265*, 2024.

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

[18] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024.

[19] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024.

[20] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*, 2024.

[21] Hassan Mehmood, Ekaterina Gilman, Marta Cortes, Panos Kostakos, Andrew Byrne, Katerina Valta, Stavros Tekes, and Jukka Riekki. Implementing big data lake for heterogeneous data sources. In *2019 ieee 35th international conference on data engineering workshops (icdew)*, pages 37–44. IEEE, 2019.

[22] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12): 1986–1989, 2019.

[23] Mohammadreza Pourreza and Davood Rafiei. Evaluating cross-domain text-to-sql models and benchmarks. *arXiv preprint arXiv:2310.18538*, 2023.

[24] Mohammadreza Pourreza and Davood Rafiei. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*, 2024.

[25] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. *arXiv preprint arXiv:2405.15307*, 2024.

[26] Dezhao Song, Frank Schilder, Shai Hertz, Giuseppe Saltini, Charese Smiley, Phani Nivarthi, Oren Hazai, Dudi Landau, Mike Zaharkin, Tom Zielund, et al. Building and querying an enterprise knowledge graph. *IEEE Transactions on Services Computing*, 12(3):356–369, 2017.

[27] Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. Exploring chain-of-thought style prompting for text-to-sql. *arXiv preprint arXiv:2305.14215*, 2023.

[28] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[29] Shuo Zhang, Zezhou Huang, and Eugene Wu. Data cleaning using large language models. *arXiv preprint arXiv:2410.15547*, 2024.