Ship Marine Strategy Database Access using Natural Language: An application of LLM-based Text-to-SQL model

by

Arunoday Ghorai

Under the Supervision of Dr Vikram Goyal

Indraprastha Institute of Information Technology Delhi

December, 2024

Ship Marine Strategy Database Access using Natural Language: An application of LLM-based Text-to-SQL model

by

Arunoday Ghorai

Submitted

in partial fulfillment of the requirements for the degree of

Master of Technology

to

Indraprastha Institute of Information Technology Delhi

December, 2024

Certificate

This is to certify that the thesis titled "Ship Marine Strategy Database Access using Natural Language: An Application of LLM Based Text-to-SQL Model" being submitted by Arunoday Ghorai to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

December, 2024

Vikram Goyal
Department of Computer Science Engineering
Indraprastha Institute of Information Technology Delhi
New Delhi 110 020

Abstract

The growing reliance on relational databases across industries and the ability to efficiently query and extract from a structured database has become a crucial skill in the industry. However, the Complexity of SQL Syntax creates a barrier for non-technical uses limiting their ability to interact with databases effectively. Natural Language to SQL (NL-to-SQL) query generation performs a critical task in bridging gap between non-technical users and relational databases and enables intuitive data interaction with out any need for SQL expertise. This thesis first explores various Text-to-SQL approaches, leveraging both proprietary model like Open AI's GPT-4 and open-source models like RESDSQL, focusing on their performance across benchmark datasets like Spider, CoSQL and SPARC. Additionally, two datasets, MORD and CMEC are prepared from the real world use cases to highlight unique challenges such as hierarchical data structures, string matching operations, and privacy issues. The MORD dataset was queried using GPT-4 integrated with LangChain, to showcase natural language interaction with data and the usability of proprietary models without any tuning to domain specific dataset. Meanwhile the CMEC dataset is a privately curated dataset and access to it needs to be confidential. So we use open source models like RESDSQL that run on local server in order to minimize leakage. The dataset is pre-processed into a relational schema, and RESDSQL is fine tuned on curated NL to SQL pairs to improve performance. String matching techniques are applied to prepare better prompts in order to further enhance the results generated by the model.

Acknowledgements

I am grateful to many individuals and institutions whose support and encouragement have been indispensable throughout my journey in completing this thesis. First and foremost, I am deeply indebted to Dr. Vikram Goyal for providing me with the remarkable opportunity to work on an industrial project. His guidance, unwavering support, and encouragement have been invaluable in the completion of this thesis.

I extend my heartfelt gratitude to my PhD mentor, Pankaj Kumar, for his invaluable suggestions, support, encouragement, and guidance throughout this work. Furthermore, I would like to express my sincere thanks to my parents and my brother for their constant motivation and support during this period.

I am also thankful to my invaluable friends, whose moral and emotional support have been instrumental in the successful completion of this thesis.

My MTech thesis journey began in January 2023 and was marked by numerous personal and academic challenges. The process involved extensive literature review, benchmarking state-of-the-art models, creating a custom dataset, fine-tuning, and improving the model on the dataset. Despite these hurdles, the journey has been incredibly rewarding. One of the major milestones was securing a placement at Chegg India.

Thank you all for being a part of my journey.

Arunoday Ghorai

Contents

List of Figures

List of Tables

1. Introduction

1.1 Background

SQL is one of the standard programming languages used for management and interaction with relational databases. It is one of the key technologies that enables integration with modern technologies like big data, cloud computing, and hybrid data models. Recent innovations in the field of Machine Learning, particularly Large Language Models (LLMs), have opened avenues to automate various aspects of SQL generation.

Among such advancements, Text-to-SQL emerged as a significant approach within Natural Language Processing (NLP) to convert natural language queries (NLQs) into executable SQL statements. This approach bridges the gap between technical users and relational databases, enabling intuitive interaction with data without any need for SQL expertise. Recent breakthroughs were driven by fine-tuned Pre-Trained Language Models, which set new standards for benchmark datasets such as Spider, CoSQL, SParC, and BIRD. These benchmarks provide diverse, challenging environments for the evaluation of Text-to-SQL systems, pushing boundaries of generalization and accuracy.

In this project, we aim to explore various approaches to automate database querying with both open-source and proprietary models. Then we will evaluate the strengths and limitations, particularly focusing on the performance across the standard benchmarks. Based on the findings, the most effective model will be selected and deployed to solve a real-world datasets as stated in the next section.

1.2 Problem Definition

1. **MORD Dataset:** In this problem, we were given a sample Excel dataset that contained various details related to the development of roads/projects in various states and Union Territories. The dataset was open source, and as such, there were no issues regarding privacy. A Natural Language to Query Interface needs to be developed using both proprietary or open-source models to query the dataset and return relevant data based on the user query.

2. **CMEC Dataset:** In this problem, we were given an Excel dataset that contained various details related to the Indian port administration and the bilateral relations with various countries and the various pacts. The dataset is not a publicly available dataset, and as such, the entire system shall be built with privacy-aware methodology in order to eliminate the leakage of data in any form. Using natural language queries, the dataset needs to be queried and should return relevant results in a tabular or SQL result.

2. Literature Survey

The literature review for this work has been divided into two sections:- One for surveying various Text-to-SQL datasets and the other for the State Of The Art models trained and evaluated on these datasets.

2.1 Text-to-SQL Datasets

Early efforts on text-to-sql domain were primarily focused on single table queries with a large scale benchmark like WikiSQL dataset [18]. However there were several limitations including handling of compositional and multi table queries and also underscored the need for schema aware representations and cross domain generalization. This catalysed the creation of new datasets like Spider [16], CoSQL [15] and SPARC [17], which significantly influenced evolution of Text-To-SQL researcher.

Spider [16] dataset introduced complex multi table queries and cross domain evaluation, which require models to generalize unseen schemas. Spider consists of 10,181 natural language questions paired with 5,693 unique complex SQL queries across 200 databases spanning 138 domains. The databases included multi-table schemas with an average of 27.6 columns and 8.8 foreign keys per database which ensured diversity in structure and content. Questions were carefully crafted to reflect realistic database querying needs which covered a wide range of SQL operations such as JOINs, GROUP BY, HAVING, and nested queries. Spider's design introduced a novel cross-domain evaluation setting where the training and test datasets have disjoint schemas. This design prevents models from memorizing schema-specific patterns, forcing them to generalize new databases and queries. The queries included a wide variety of SQL operations and components, such as nested subqueries, aggregations, as well as set operations like INTERSECT, UNION and EXCEPT. The dataset SQL queries was categorized into four difficulty levels: easy, medium, hard, and extra hard, based on the complexity of Operations. Three new evaluation metrics were added in Spider namely Component Matching, Exact Matching, and Execution Accuracy which evaluated models' ability to generate SQL components correctly, produce queries equivalent to ground truth, and execute queries that return correct results. Component matching assess the semantic correctness of individual SQL query components to provide detailed insights into a model's strength and weaknesses.

[16]

Several challenges were introduced by spider dataset including schema generalization where models need to adapt to unseen database schemas during testing, requiring robust schema linking and representation techniques. Also, there were lot of Complex Queries which involved multi table joins, nested queries and advanced SQL clauses and also Spider's natural language questions [16] required precise schema linking and context aware SQL generation that added more challenge to Spider dataset. The cross domain learning made it the de facto benchmark for Text-to-SQL for pre-trained transformer models like T5 and GPT-3.

Spider, [16] however, is a single turn dataset where each query is treated as an isolated instance. Real world database interactions may have an iterative nature where users can refine elaborate or clarify questions over multiple turns. An absence of conversational context meant models trained on Spider dataset cannot maintain a dialogue history. CoSQL [15] extended Spider by introducing multi-turn, conversational queries where each dialogue simulates an interactive scenario where users can refine their queries based on system responses, forcing models to maintain dialogue state tracking and keeping prior context while generating SQL queries. CoSQL [15] also incorporates ambiguous and unanswerable queries which requires models to identify those cases and respond appropriately by generating clarifications. It introduces the task of generating natural language responses for SQL results, which helps users understand the data returned by SQL queries and requires systems to clarify ambiguous queries by asking relevant follow-up questions.

Like CoSQL, SParC [17] dataset was built upon the foundational work established by Spider dataset. SParC incorporated context-dependent multi-turn interactions which enhanced semantic diversity. Each question sequence is structured around an interaction goal, such as summarizing data, exploring patterns, or answering complex queries. SParC presented a multi-turn SQL query generation like CoSQL where later queries depend on the logical structure and constraints established in earlier turns. Later turns may introduce additional SQL components such as GROUP BY, ORDER BY, and nested queries which challenged models to expand the logical structure progressively. New evaluation metrics such as interaction level accuracy was introduced where models must correctly answer all questions in a turn in a sequence to succeed.[17]

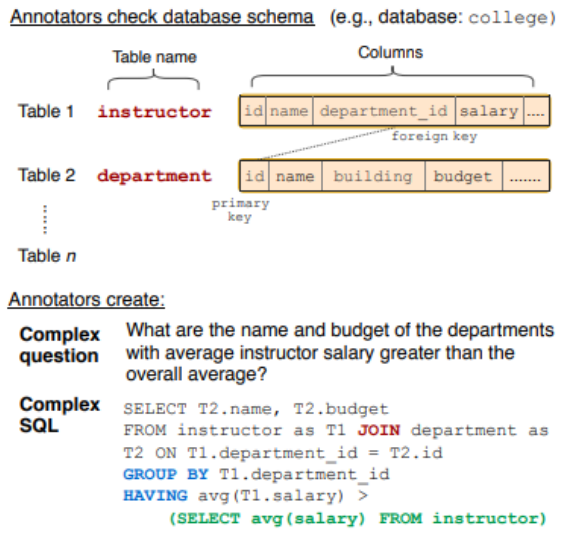

Figure 1: Spider example complex question containing SQL of joining of multiple tables, a GROUP BY component and nested query [16]

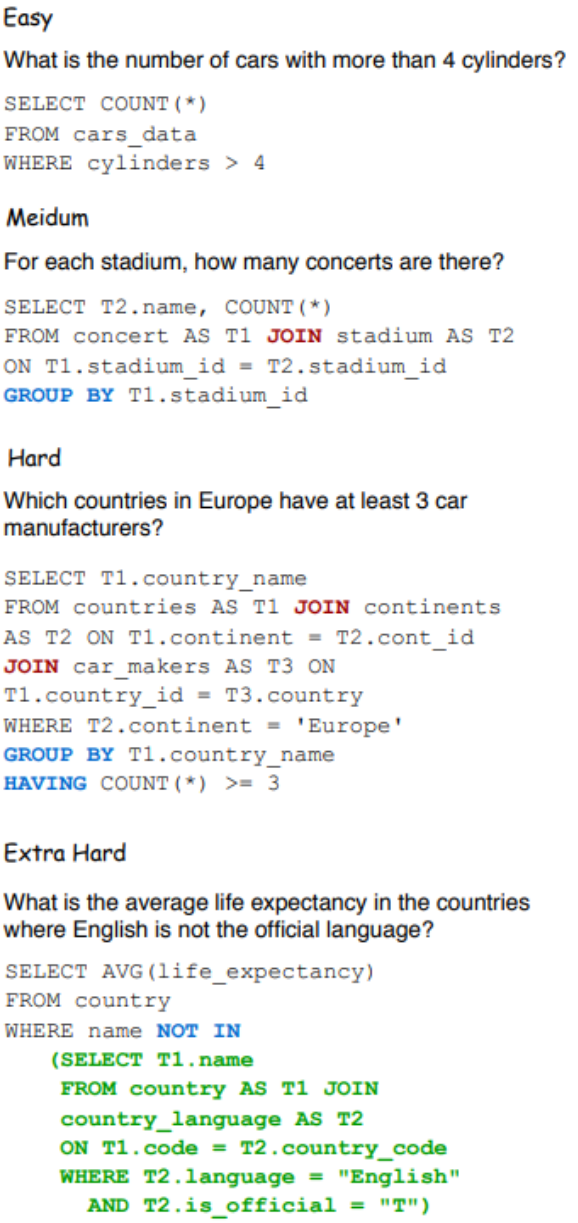

Figure 2: Spider SQL Query Example in 4 hardness levels [16]

Domain Knowledge was added to the Spider dataset in Spider-DK as a core evaluation component. Spider DK curates examples which requires and understanding of domain-specific knowledge with implicit column references, synonyms and logical references. For example, A query asking for "American airlines" may require understanding that "American" is a synonym for "USA." Models must correctly infer this relationship when generating SQL queries. This enhancement tests models ability to generalize domain knowledge to unseen queries. It also includes query in natural language to evaluate the robustness of the models to linguistic variability to ensure that the models are based on

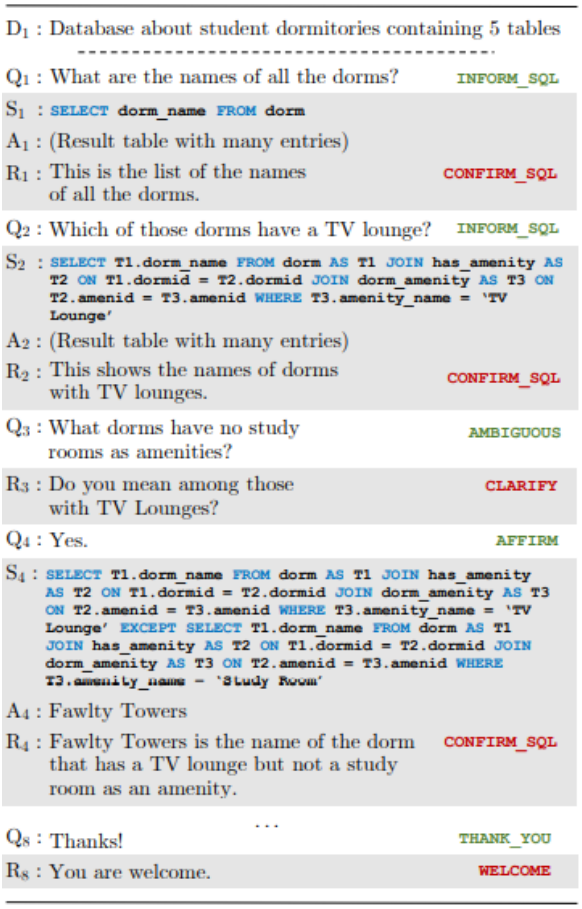

Figure 3: CoSQL Dataset Dialogue Structure [15]

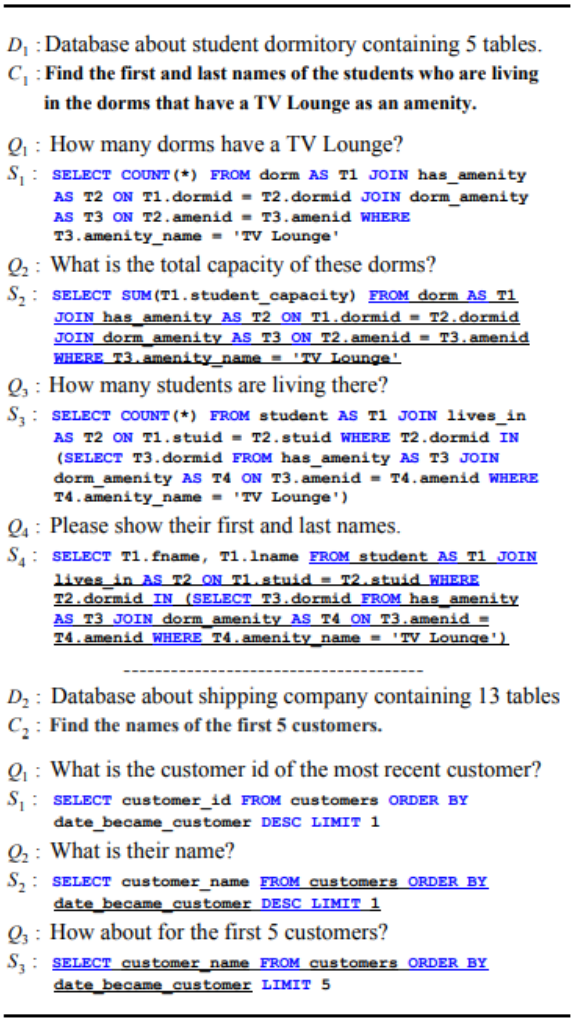

Figure 4: SParC Dataset Dialogue Structure [15]

semantic understanding rather than surface level patterns. Several domain knowledges introduced by Spider-DK include Implicit Column Mentions, simple logical inferences, synonym substitutions Boolean like conditions and dynamic conflict resolution. Spider Dk evaluates model performance on metrics tailored to domain knowledge to highlight areas of model failure due to semantic or logical mismatches rather than schema linking issues, offering a more realistic assessment of cross-domain Text-to-SQL generation.

Spider Syn introduces synonym substitution to test models robustness and semantic understanding as spider assumed that Natural Language questions explicitly referenced schema items which is unrealistic in real world settings. Models trained on Spider originally showed poor performance when paraphrase or synonymic variations of schema references which reduces robustness in real world scenarios. Spider-Syn replaces schema related words in NL questions with manually curated synonyms to ensure that SQL

queries remain consistent while introducing linguistic variability. For example an NL question in Spider referencing "course" might be modified to use "curriculum" in Spider-Syn. This allows to evaluate models ability to link schema item indirectly referenced through synonyms.

Spider Realistic builds on Spider, introducing realistic challenges by focusing on text-table alignment and schema grounding. Modifies NL questions to remove explicit references to column names, replacing them with descriptive phrases or paraphrases. For example, a question in Spider "Find the number of concerts happened in the stadium with the highest capacity." Can be paraphrased to "Find the number of concerts happened in the stadium that can accommodate the most people." By removing explicit column mentions, semantic understanding and schema inference is better emphasized requiring models to link descriptive NL phrases to the correct schema components. The NL questions are manually revised to ensure realistic phrasing while preserving SQL query structure from original dataset. The dataset allows for Robustness to various Real World Scenarios. For the purpose of this Thesis, we will only consider the above datasets for evaluation of models.

Notable Mentions:- The BIRD dataset (BIg bench for laRge-scale Database grounded in Text-to-SQL tasks) was introduced to focus primarily on database schema with limited consideration of database values and real world complexities. BIRD emphasizes database value comprehension with external knowledge reasoning and SQL execution efficiency. It integrates Numeric Reasonin, Domain Knowledge, Synonym substitution and Value Illustration. BIRD introduces Valid Efficiency Score evaluating both the correctness and execution efficiency of generated SQLs, which is measured by runtime, throughput and memory cost.

	Only Modify the NL
Spider	... *in the order of birth date.*
Spider-DK	... *order of their birth date from old to young.*
	Modify both NL and SQL
Spider	*Compute the average age of dogs.* `select avg(age) from dogs`

Spider-DK	*Compute the average age of abandoned dogs.*
	`select avg(age) from dogs`
	`where abandoned_y = 1`

Figure 5: Spider-DK modifications To Spider [5]

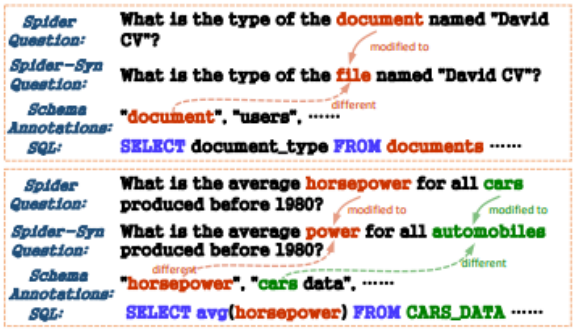

Figure 6: Spider-SYN modifications To Spider [4]

Example	Type
Show name, country, age for all singers ~~ordered by age~~ from the oldest to the youngest.	Remove
Find the number of concerts happened in the stadium ~~with the highest *capacity*~~ that can accommodate the most people. How many pets ~~have a greater *weight* than 10~~ are over 10 lbs?	paraphrase

Figure 7: Spider-Realistic modifications To Spider [2]

2.2 Text-to-SQL Models

Earliest models like TypeSQL [14] was designed to explore schema linking in Text-to-SQL tasks, combining traditional neural networks with schema features. This approach was refined further with release of BERT (Bidirectional Encoder Representations from Transformers) [3], introducing pre-trained contextual embeddings which significantly improves schema alignment. Sequence-to-sequence transformer models, such as T5 (Text-to-Text Transfer Transformer) [9] was pre-trained on diverse text-to-text tasks, was fine-tuned on datasets like Spider and WikiSQL to generate SQL queries directly from NLQs. RAT-SQL (Relation-Aware Transformer) (Wang et al., 2020)[13] further extended transformer capabilities by introducing relation-aware attention mechanisms which explicitly models schema relationships and significantly improves the performance on Spider and related datasets as shown in Figure 8.

`RAT-SQL` with BERT struggled to accurately detect explicitly mentioned columns or infer columns from cell values, leading to incomplete or inaccurate SQL queries. `GAP` [12] introduced **Column Prediction** and **Column Recovery**, which explicitly train

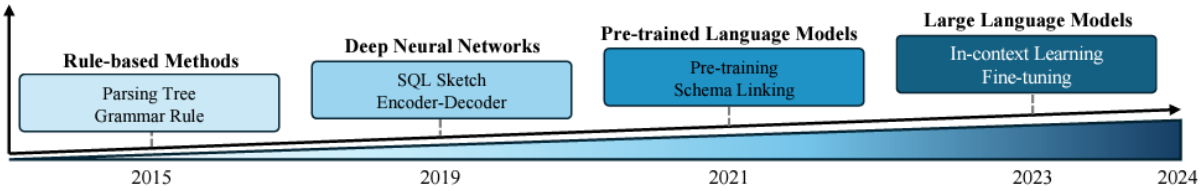

Figure 8: Evolution of Text-to-SQL approach over time [6]

the model to identify relevant schema columns, while the latter enables the model to infer columns based on contextual information such as database cell values. It directly overcomes schema-linking challenges.

`GAP` [12] uses the SQL Generation task, which trains the model to construct queries involving nested subqueries, `HAVING` clauses, and multi-table joins iteratively to better handle its compositional and syntactic complexities. `GAP` leverages high-quality synthetic data generated through SQL-to-Text and Table-To-Text generation to limit reliance on indirect schema and language alignment, thereby eliminating dependency on noisy pre-training data. These enhancements allowed `GAP` to achieve state-of-the-art results on **SPIDER** and improved compositional generation.

However, `GAP` [12] is reliant on synthetic data, limiting its robustness for edge cases present in real-world scenarios. Additionally, domain-specific tasks with schemas or query structures not covered in benchmarks like **SPIDER** might require further fine-tuning. `GAP`[12] also faces challenges in multi-turn interactions and maintaining contextual coherence across queries. It struggles to maintain accuracy in scenarios involving conversational SQL tasks, such as those present in the **SParC** [17] and **CoSQL** [15] datasets, which require tracking user intent and incorporating the context of previous turns.

`RASAT` [8] addresses these shortcomings by incorporating relational structures directly into a pre-trained T5 model through a **relation-aware self-attention mechanism**. It leverages a broader range of relations, including schema encoding, syntactic dependencies, and coreference resolution, allowing `RASAT` [8] to handle complex conversational scenarios, dynamically integrating multi-turn interaction context. `RASAT` [8] augments T5's self-attention with relational embeddings, thus achieving state-of-the-art results on benchmarks **SParC** [17] and **CoSQL** [15], while improving interaction-level accuracy (IEX).

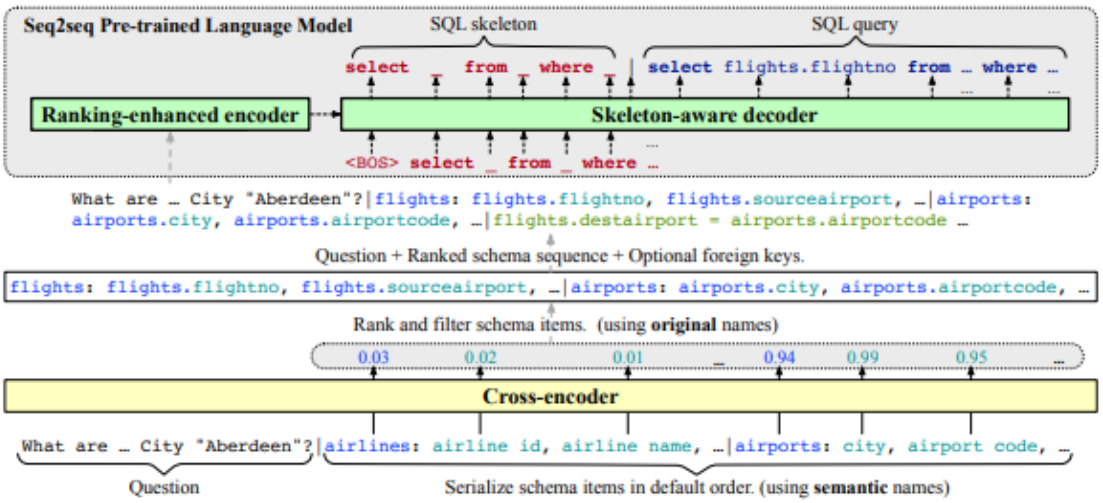

Figure 9: Overview of RESDSQL Framework [7]

OOD (Out of Distribution) Generalization [10] techniques tackled compositional and domain generalization challenges that persisted in models like GAP [12] and RASAT [8]. They employed two impactful methods: **Token Pre-Processing** and **Component Boundary Marking**. *Token Pre-Processing* ensured that schema items and SQL keywords were split into meaningful tokens by preserving their semantic boundaries. For example, converting 'avg' to 'average' and 'pet_age' to 'pet_age' ensured that the model could leverage its pre-trained knowledge effectively, allowing OOD to overcome struggles faced by earlier models in fragmented tokenization of schema components. *Component Boundary Marking* introduced special tokens to define aligned components between NL and SQL, which is beneficial for compositional generalization. This enables the model to understand complex SQL structures involving nested queries and multiple table joins. OOD uses explicit boundary markers to create clear semantic mappings, reducing ambiguity in query generation. This approach provided simplicity and effectiveness in improving domain and compositional generalization without the need for additional complex architectures or extensive task-specific training. It demonstrated notable gains in **EM** and **EX** on **SPIDER**. However, limitations were noticed in schema linking and SQL skeleton parsing for complex queries involving nested structures, multiple joins, and advanced SQL operators.

RESDSQL (Ranking-enhanced Encoding plus a Skeleton-aware Decoding framework for Text-to-SQL) [7] introduces a novel approach to Text-to-SQL parsing by decoupling schema linking and skeleton parsing. RESDSQL ranks schema items first through a

cross encoder, which identifies and filters relevant tables and columns. These schema items are then passed to the encoder, significantly reducing the noise and enhancing schema linking accuracy. Parallelly the skeleton aware decoder generates a SQL skeleton as intermediate step before constructing the full query. The ranking enhanced decoder is a key feature in RESDSQL using semantic names and a column enhanced layer to improve schema linking in cases where table names were not explicitly mentioned in query. This encoding framework ensures inclusion of essential schema items while minimising irrelevant noise. SQL queries are normalized by skeleton aware decoder, generating skeletons which guide subsequent SQL query formation, improving results on complex and nested SQL queries. RESDSQL achieves state of the art performance on the Spider dataset and its robustness variants like Spider-DK, Spider-Syn and Spider-Realistic. It outperforms models like RAT-SQL, RASAT, OOD Generalization, and GAP Text-to-SQL in Spider's Development Set. Model also demonstrates exceptional robustness to schema perturbations and question paraphrasing, highlighting its generalization across diverse Text-to-SQL challenges. [7]

Models	**Spider**		**Spider SYN**		**Spider Realistic**		**Spider DK**		**COSQL**		**Sparc**	
	EM	**EX**	**EM**	**EX**	**EM**	**EX**	**EM**	**EX**	**EM**	**EX**	**EM**	**EX**
OOD Generalization	0.71	0.75	-	-	-	-	-	-	-	-	-	-
GAP Text2Sql	0.72	-	-	-	-	-	-	-	-	-	-	-
RASAT	0.72	0.75	-	-	-	-	-	-	0.56	0.63	0.63	0.67
RESDSQL-Base	0.71	**0.77**	0.58	0.66	0.65	0.69	0.44	0.56	-	-	-	-
RESDSQL-3B	0.78	**0.81**	0.69	0.76	0.77	0.81	0.53	0.66	-	-	-	-
RESDSQL-large	0.75	**0.80**	0.62	0.69	0.69	0.72	0.51	0.61	-	-	-	-

Table 1: Performance Comparison of Text-to-SQL Models

3. Methodology

3.1 Proof of Concept for Natural Language Querying with MORD Dataset

Large Language Models (LLMs) can be categorized into two types: proprietary models, such as OpenAI's GPT family, Google's Gemini, and Anthropic's Claude, which are renowned for their state-of-the-art performance, and open-source models, such as Meta's LLaMA family and Mistral models, which offer greater flexibility for research and applications.

While proprietary models deliver high performance, they are often restricted by access limitations, licensing costs, and a lack of customization options. In contrast, open-source models can be fine-tuned and deployed without constraints imposed by APIs and server-related issues. However, both types of models are generally pre-trained on large and diverse datasets and are not optimized for specific tasks like Text-to-SQL.

Proprietary models cannot be fine-tuned directly and thus require frameworks like ReAct (Reasoning and Acting) to adapt them for Text-to-SQL tasks. ReAct frameworks, often referred to as agentic frameworks, enhance models' ability to reason, interact with databases, and generate accurate queries by combining reasoning capabilities with external tools and datasets.

For this study, we focused on using a publicly available dataset and OpenAI's GPT-4 API. The dataset, initially in raw format, was pre-processed and converted into a CSV file to enable efficient querying. This structured format works well within the capabilities of LLMs when paired with tabular data handling frameworks.

To facilitate the interaction between the dataset and the GPT-4 model, the LangChain framework was used. LangChain simplifies the development of applications that leverage LLMs as core components. It provides utilities to integrate natural language processing with external tools and data sources. Specifically, the `create_csv_agent()` function provided by LangChain was utilized to establish a seamless interface for querying CSV files.

The CSV-formatted dataset was loaded into the `create_csv_agent()` utility along with the GPT-4 model. The CSV agent serves as a mediator, translating natural language

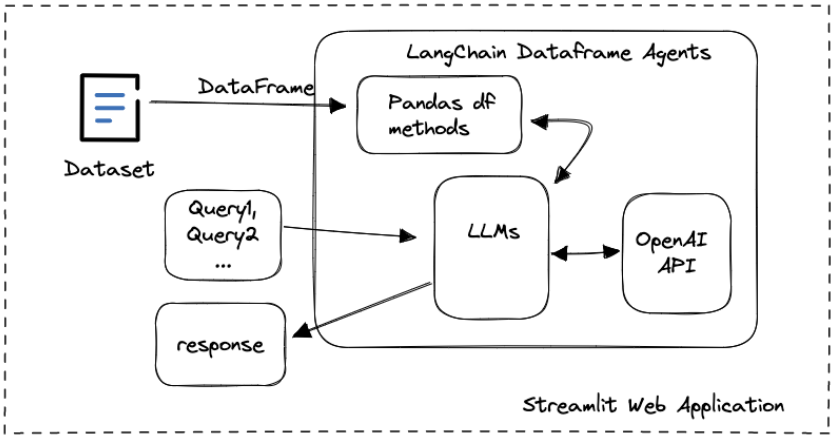

Figure 10: LangChain Methodology [1]

queries into structured operations on the dataset. For example, a query like:

> *"What is the number of activities in ongoing projects where the plan is finalized in the state Uttarakhand?"*

is parsed and executed on the dataset, and the resulting value is returned to the user.

This approach highlights the potential of proprietary LLMs like GPT-4 for Text-to-SQL tasks, even without domain-specific fine-tuning. Using LangChain's abstraction, the complexities of designing custom parsers for SQL generation logic were avoided, allowing us to focus on retrieving insights from the dataset. The combination of GPT-4 and LangChain bridges the gap between unstructured natural language queries and structured dataset operations, demonstrating utility in real-world scenarios.

Limitations: While GPT-4 delivers high accuracy for straightforward queries, more complex operations that may involve nested queries, multi-table joins, or intricate filtering may pose challenges. Reliance on proprietary APIs introduces dependencies in terms of scalability and cost of deployment. Open-source alternatives can address these limitations by providing more cost-effective and adaptable solutions for domain-specific datasets, further enhancing accuracy, generalizability, and ease of deployment. Also risks related to data leakage can also lead to privacy issues in proprietary models like GPT.

3.2 Natural Language to SQL Query Generation with CMEC Dataset

3.2.1 Description of CMEC Dataset

The CMEC (Centre for Maritime Economy and Connectivity) A1_A2 dataset is a comprehensive resource developed to analyze and support India's strategic and maritime initiatives under the Maritime Amrit Kaal Vision 2047 (MAKV-2047). The dataset contains relevant details of international bilateral ties, infrastructure development, and technological advancement in maritime commerce and connectivity.

It encapsulates national-level interventions, key performance indicators (KPIs), and descriptions of maritime goals to evaluate India's future maritime economic development strategies. Structurally, the dataset adopts a hierarchical format categorizing data into three main levels: A0, A1 goals, and A2 interventions. The A0 layer defines overarching themes and objectives, which further cascade into A1 goals, capturing strategic national-level aspirations. These goals then translate into actionable A2 interventions, detailing specific operational strategies. Each intervention is further classified under Vibhas Navic, highlighting the programmatic framework governing these initiatives.

This hierarchical organization allows for seamless mapping of broader goals to their specific implementations. The dataset schema incorporates critical columns, including Vibhas TM Navic, Type (National Level), Description (National Level), KPIs, Implementing Agencies, Categories, MoPSW wing, and MIV/MAKV Reference.

3.2.2 Preprocessing of the Dataset

The dataset was broken down into six tables: `a0`, `a1_goals`, `a2_interventions`, `KPIs`, `a0_a1`, and `a1_a2`, while preserving its hierarchical structure. The `a0` table entries contain the type a0, while `a1_goals` and `a2_interventions` correspond to types A1 and A2 respectively. We found that KPIs were being governed by `a1_goals`, many of which were missing, and as such they were extracted into a separate `KPIs` table which was linked with the `a1_goals` table through a foreign key referencing the `code` column. To maintain hierarchical relationships, two bridging tables were created: `a0_a1`, linking the `code` column from `a0` and `a1_goals`, and `a1_a2` linking `a1_goals` and `a2_interventions`. Certain column names were renamed, such as "Vibhas Navic" to "strategic category" and

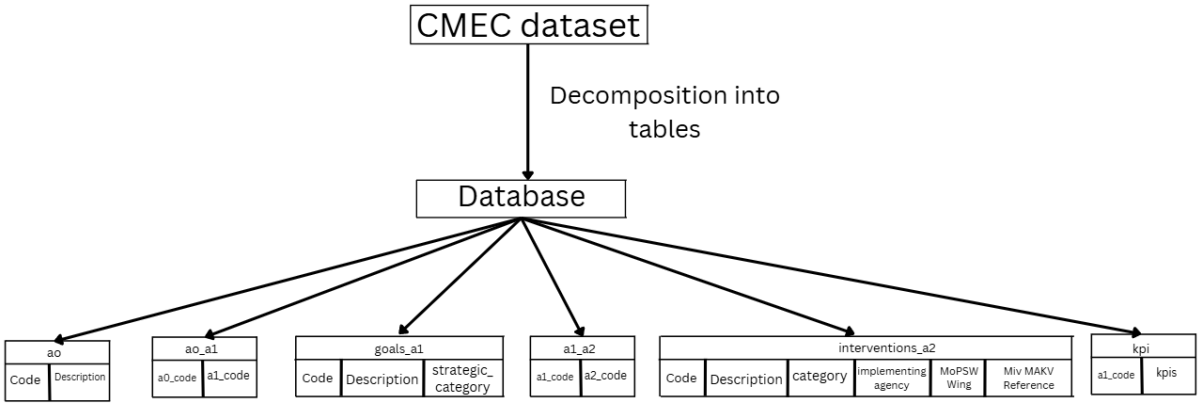

Figure 11: Decomposition of the Cmec dataset into a database containing the tables

"Implementing Agency — TM Organization" to `implementing_agency`. Additionally, table and column names were normalized by replacing blank spaces, parentheses, and commas with underscores (_). The dataset is uploaded to a PostgreSQL database, and the schema is saved for further operations. To enhance and validate the usability of Large Language Models in our database, we developed a robust Natural Language Query (NLQ) to SQL dataset containing NLQs with their corresponding SQL query pairs. The queries were both handwritten and augmented using ChatGPT by modifying the prompt to include the database schema iteratively. ChatGPT was instructed specifically to generate NL-SQL pairs by taking inspiration from the SPIDER dataset structure, which takes into account the hierarchical nature and schema constraints of our CMEC dataset. This resulted in 102 queries for the training set and 49 queries for the development set. In addition to being manually tested for correctness, accuracy, and functionality, the NL-SQL pairings were also examined to ensure they ran properly, had accurate outputs, and returned non-empty results. The scope of the questions ranged from simple SQL queries like `SELECT` and `WHERE` clauses to more intricate queries that involved advanced SET procedures like `UNION` and `INTERSECT`, as well as numerous joins across many tables. Additional SQL functions, such as `LENGTH`, were incorporated into the queries for text-based operations because the majority of the data in the database is textual. String matching queries were also created to ensure that records could be retrieved from the database. These queries used SQL's `LIKE` operator to retrieve results when certain descriptors matched a substring of text. For example, an NL query could be "Give me all those interventions corresponding to Europe West" which was translated into an SQL query using the pattern `LIKE '%Europe West%'`. This introduces an additional

layer of complexity for NL-SQL models like RESDSQL, [7] as the SPIDER dataset does not explicitly cover such substring-based retrieval results, even though the dataset and evaluation metrics in SPIDER contain operations like the `LIKE` operator. This makes the dataset particularly challenging and novel, providing an additional layer for improving NL-to-SQL models.

3.2.3 Model Development

`RESDSQL`, [7] the current state-of-the-art Text-to-SQL model, tops the benchmarks in SPIDER as found in Table 1. However, despite its strong performance on the SPIDER dataset, `RESDSQL` struggled to generalize effectively on our own custom CMEC dataset, achieving a mere 20% on the development set. It can be said that the model is not adaptable when faced with a new database schema and unseen queries, especially those requiring string-matching operations and hierarchical table joins. For this purpose, we proposed fine-tuning `RESDSQL` on the curated training set for a few epochs using the existing training script. Checkpoints were saved during training and were evaluated on the development set, with the best checkpoint selected for inference. The results are discussed in the Results section.

Additionally, we implemented a string-matching enhancement script to provide additional context to the model. A new auxiliary dataset was created using columns identified as critical for string-matching operations, namely: `type`, `descriptions`, and `strategic_category`. For string-matching queries, we used the BM25 [11] algorithm to identify relevant values, followed by the LCS (Longest Common Subsequence) algorithm to refine results. Instead of searching for subsequences, we modified the algorithm to find substrings. The identified values with their corresponding table names were then appended to the natural language query as additional input context. The auxiliary dataset will be contained in local system devices and will not be shared with model or anywhere else. This creates a privacy aware augmenting system that will feed extra information about the query without passing any data to the model.

For example, in the NL question, "Give me all the interventions related to Europe West," the substring `Europe West` is found in the `description` column of the `a0` table. Meanwhile, the term `interventions` refers to the `interventions_a2` table. Since there is no direct join between `a0` and `interventions_a2`, the `goals_a1` table is implicitly

Sample Queries

Question 1: How many interventions are in the `interventions_a2` table?

SQL:

```
SELECT count(*) FROM interventions_a2;
```

Question 2: Which a0 descriptions are linked to interventions in the 'Infrastructure' category?

SQL:

```
SELECT DISTINCT A.description
FROM a0 AS A
JOIN a0_a1 AS AA ON A.code = AA.a0_code
JOIN a1_a2 AS AA2 ON AA.a1_code = AA2.a1_code
JOIN interventions_a2 AS I ON AA2.a2_code = I.code
WHERE I.category = 'Infrastructure';
```

Question 3: Find all a0 descriptions linked to goals in the 'New Tech, research, startups' strategic category.

SQL:

```
SELECT DISTINCT A.description
FROM a0 AS A
JOIN a0_a1 AS A01 ON A.code = A01.a0_code
JOIN goals_a1 AS G ON A01.a1_code = G.code
WHERE G.strategic_category = 'New Tech, research, startups';
```

Figure 12: Some Example Queries of Human annotated Questions with GPT augmented Questions

required for the connection. Furthermore, the query's `WHERE` clause requires the `LIKE` operator to search for the substring `Europe West`. To address this, the augmented prompt explicitly mentions the relationships, such as the value `Europe West` belonging to the `description` column in `a0` and `interventions` matching the table `interventions_a2`.

Metrics :-For evaluation of the performance of Text-to-SQL model, we adopted metrics used in prior studies such as Exact Set Match Accuracy (EM) and Execution Accuracy (EX). [16] Exact Set Match Accuracy assesses the syntactic correctness of the generated SQL query by comparing it to the ground truth query. EM measures whether the predicted SQL query matches the ground truth exactly as a set, ignoring token order where applicable. This ensures that queries with equivalent semantics but slightly different formatting are still considered accurate. However, EM may penalize syntactically valid queries that deviate from the ground truth due to minor variations, even if they produce correct results. Execution Accuracy (EX) evaluates the semantic correctness of the generated query by executing both the predicted query and the ground truth query on the database and comparing their outputs. This metric accounts for the fact that there can be multiple syntactically different but semantically equivalent SQL queries for the same natural language question. By focusing on the execution results, EX captures the true utility of the generated query in real-world scenarios.

4. Results

Method	EM	EX
Without Finetune	0.204	0.34
With Finetune	0.46	0.77
With Finetune + Additional Information Included in Prompt	0.55	0.75

Table 2: Results on Finetuning RESDSQL-Base

The results are summarized in Table 2, with the exact match metrics (EM) and execution accuracy (EX) in the curated dataset. The baseline `RESDSQL` model without any fine-tuning performed poorly, with a meagre 20% EM and 34% EX accuracy on our development set, highlighting potential adaptability issues with the `RESDSQL` model on our dataset. However, after fine-tuning the model on our training set, we observed a significant improvement in the EX metric, with better alignment towards handling queries in this domain.

String matching techniques further enhanced performance, although with a smaller improvement in EM accuracy. This enhancement was due to the effect of better generalization of table names and improved schema filtering, ensuring that the correct columns and values were linked to their respective tables. The string matching script allowed the model to infer relationships and apply table joins with more accuracy.

One of the standout features of this approach is its low computational cost. The primary searching mechanism employs the BM25 algorithm, which can efficiently retrieve relevant results. Building on this, the LCS (Longest Common Subsequence) algorithm is utilized to refine the search results. While the LCS algorithm theoretically has a time complexity of $O(m \times n)$, the practical implementation operates on a much smaller search space that reduces the computational cost to a negligible $O(1)$ in this scenario, making it highly efficient for deployment. Additionally, it provides significant advantages in terms of privacy and security. By using an auxiliary dataset stored on host/local systems, the method ensures that no sensitive data is passed to the language model. This approach prevents potential data leakage and also provides the model with enough contextual information to generate SQL queries more accurately.

5. Conclusion and Future Work

This work explored the challenges and solutions for natural language (NL) to SQL query generation, focusing on both proprietary and open-source models. Proprietary models like GPT-4 demonstrated strong capability to handle straightforward queries when integrated with frameworks such as LangChain. However, several limitations were observed in the execution of complex SQL operations.

In contrast, open-source models like RESDSQL provide greater flexibility for domain-specific tasks, though significant adaptability issues were noted on custom datasets without fine-tuning. The investigation into the CMEC dataset posed significant real-world challenges, including hierarchical data structures, privacy concerns, and string-matching operations. The pre-processing steps were crucial for structuring the data effectively.

Fine-tuning the RESDSQL model on the curated training set resulted in notable improvements in both Exact Match (EM) and Execution Accuracy (EX), demonstrating the importance of domain-specific optimization. The addition of string-matching techniques enhanced query accuracy by improving schema linking and table relationships, addressing gaps not covered by standard datasets like Spider.

Despite advancements, challenges remain in scaling solutions to more complex datasets and further improving generalization to unseen schemas. Future work can focus on using open-source models like Llama to further improve accuracy on benchmark datasets through data augmentation techniques, as well as employing enhanced versions of string-matching techniques.

References

[1] DataDrivenInvestor. Create a stock chatbot with your own csv data. https://medium.datadriveninvestor.com/create-a-stock-chatbot-with-your-own-csv-data-0d11918ee0b3, 2023. Accessed: 2024-12-19.

[2] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. Structure-grounded pretraining for text-to-sql. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[4] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of text-to-sql models against synonym substitution, 2021.

[5] Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-sql generalization, 2021.

[6] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql, 2024.

[7] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql, 2023.

[8] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql, 2022.

[9] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[10] Daking Rai, Bailin Wang, Yilun Zhou, and Ziyu Yao. Improving generalization in language model-based text-to-sql semantic parsing: Two simple semantic boundary-based techniques, 2023.

[11] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009.

[12] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training, 2020.

[13] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers, 2021.

[14] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation, 2018.

[15] Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter S Lasecki, and Dragomir Radev. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases, 2019.

[16] 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. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019.

[17] Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. Sparc: Cross-domain semantic parsing in context, 2019.

[18] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017.