MultiSQL: A Schema-Integrated Context-Dependent Text2SQL Dataset with Diverse SQL Operations

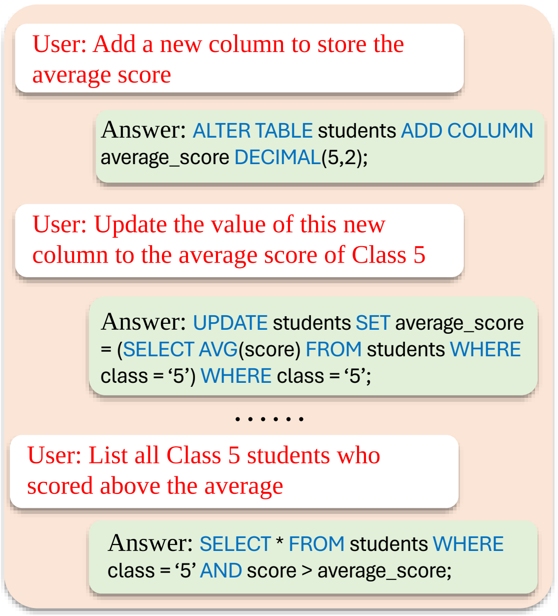
Chunhui Li[1,2] **Yifan Wang**[1,2] **Zhen Wu**[1,2*] **Zhen Yu**[3]
Fei Zhao[1,2] **Shujian Huang**[1] **Xinyu Dai**[1,2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
[3]Tencent
{lich, wangyifan, zhaof}@smail.nju.edu.cn, zhenzyu@tencent.com
{wuz, huangsj, daixinyu}@nju.edu.cn

Abstract

Text2SQL is a task that translates natural language into SQL statements. Context-dependent Text2SQL offers a more natural database interaction by simulating dialogues between users and databases, with CoSQL and SparC as representative datasets. Yet, these datasets struggle to accurately replicate real-world situations. To address this, we introduce MultiSQL, which extends them in three key aspects: (1) Diverse SQL Operations. We incorporate diverse SQL types such as Create, Update, and Insert to broaden the scope of SQL operations. (2) Schema-Integrated Context. We integrated query context with database schema dependencies to better depict database complexity. (3) Extended Dialogues. We expand dialogue length to better simulate long conversations and complex interactions. This multi-type, schema-integrated, context-dependent Text2SQL dataset comprises nearly 800 dialogue groups and over 9,000 interaction turns across 166 complex databases, offering a better benchmark for interactive user-database dialogue. Addressing MultiSQL's challenges, we refined evaluation metrics to better capture diverse SQL types and schema dependencies. We designed a prompt framework that leverages historical data and self-refinement to accurately capture the dependency between text queries and database structures. Experiments with GPT-3.5, GPT-4, and LLaMA2-7B show both the effectiveness of our strategies and the challenges of MultiSQL. The datasets is available at https://github.com/grandchicken/MultiSQL.

1 Introduction

In the information age, structured data is predominantly stored in databases. We interact with them through SQL, a query language specifically designed for managing and manipulating databases. However, SQL is quite complex, requiring a deep


User: Add a new column to store the average score

Answer: ALTER TABLE students ADD COLUMN average_score DECIMAL(5,2);

User: Update the value of this new column to the average score of Class 5

Answer: UPDATE students SET average_score = (SELECT AVG(score) FROM students WHERE class = '5') WHERE class = '5';

......

User: List all Class 5 students who scored above the average

Answer: SELECT * FROM students WHERE class = '5' AND score > average_score;

Figure 1: The example of our dataset, which displays user interactions with a database in dialogue form, employing various SQL statement types.

understanding of database structures and query syntax for effective data retrieval. To address this, the Text2SQL (Katsogiannis-Meimarakis and Koutrika, 2023; Qin et al., 2022; Dong et al., 2023; Pourreza and Rafiei, 2023; Gao et al., 2023) task was proposed, allowing direct conversion of natural language into SQL queries. This enables users to interact with databases directly using natural language, greatly facilitating data access.

To assess the effectiveness of Text2SQL, several datasets have been developed. Spider (Yu et al., 2018) is the first proposed dataset in a cross-database context. It includes complex SQL types like groups, joins, and nested queries, effectively measuring model adaptability for this task. Bird (Li et al., 2023) goes further towards real-world application, incorporating noisy data and external knowledge, to accurately depict the complexity of data in real-world business scenarios, thus challenging

[*] Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2024, pages 13857–13867
August 11-16, 2024 ©2024 Association for Computational Linguistics

models in processing real-world data.

However, the above datasets assume SQL queries are stated in a single sentence, overlooking the reality that Text2SQL interactions typically occur as dialogues. In real-world scenarios, it's a more natural way for Text2SQL applications to involve users dynamically interacting with databases through dialogue. Therefore, context-dependent Text2SQL has been developed, with SparC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) serving as pivotal datasets for this task. SparC simulates the user-database interactive dialogues, while CoSQL is a corpus developed for building cross-domain, general-purpose database querying dialogue systems. In the context-dependent Text2SQL task, models must understand the context to create the right SQL statements for the current queries.

However, existing context-dependent Text2SQL datasets still have limitations in accurately replicating real-world situations. To address this, we have extended them in the following three aspects:

1. Diverse SQL Operations: Previous datasets were limited to *Select* queries. In real situations, users engage with databases not just for queries but also for modifying and managing database structures and content. Therefore, we have incorporated SQL statement types such as *Create*, *Update*, *Insert*, *Alter*, and *Delete*, making our dataset more comprehensively reflect database interactions.

2. Schema-Integrated Context: Existing context-dependent Text2SQL datasets focused solely on natural language context dependencies. However, dependencies also exist within the database schema context. For instance, as shown in Figure 1, a column added by the user through an *Alter Table* in earlier dialogue may change the structure of table schema, which is essential when it is queried later by a *Select* statement. This requires the interaction system to dynamically capture changes in the database structure during the conversation. To address this, we have integrated the dependencies between database table structures into our dataset, more accurately simulating the complexity and dynamics of real-world databases.

3. Extended Dialogues: According to statistics, existing datasets have a relatively low average number of dialogue turns. In our new dataset, we have greatly increased the number of dialogue interactions to better simulate long conversations and complex interactions in real-world scenarios.

Consequently, we propose a **Multi**-type, schema-integrated, and context-dependent Text2**SQL** dataset, called **MultiSQL** covering various types of SQL operations. It comprises nearly 800 dialogue groups, with over 9,000 turns spanning 166 complex databases. Table 1 shows its advantages compared to existing datasets.

Our dataset brings new challenges to method design and evaluation system construction. For methods, the integrated context necessitates that prediction processes thoroughly consider the semantic context generated during dialogue and changes in database structure. The presence of diverse SQL types, along with longer dialogue histories, makes it easier for models to accumulate errors during interactions. To address this, we've developed a prompt framework, integrating historical data and a self-refinement (Peng et al., 2023) mechanism to mitigate prediction errors.

For evaluation, traditional metrics include *Exact Set Match* and *Execution Accuracy*. However, for MultiSQL, the former does not account for structural changes, and the latter struggles to handle issues with non-select statements that do not return values. Consequently, we introduce *Context-Aware Match* to track dialogue-induced table structure changes, and *Database State Match* to assess SQL execution effectiveness by comparing changes of the database content.

Finally, our experiments on models like GPT-3.5 (Ouyang et al., 2022), GPT-4 (Achiam et al., 2023), and LLaMA2-7B (Touvron et al., 2023) highlight the difficulties inherent in the MultiSQL dataset and the effectiveness of our proposed approaches.

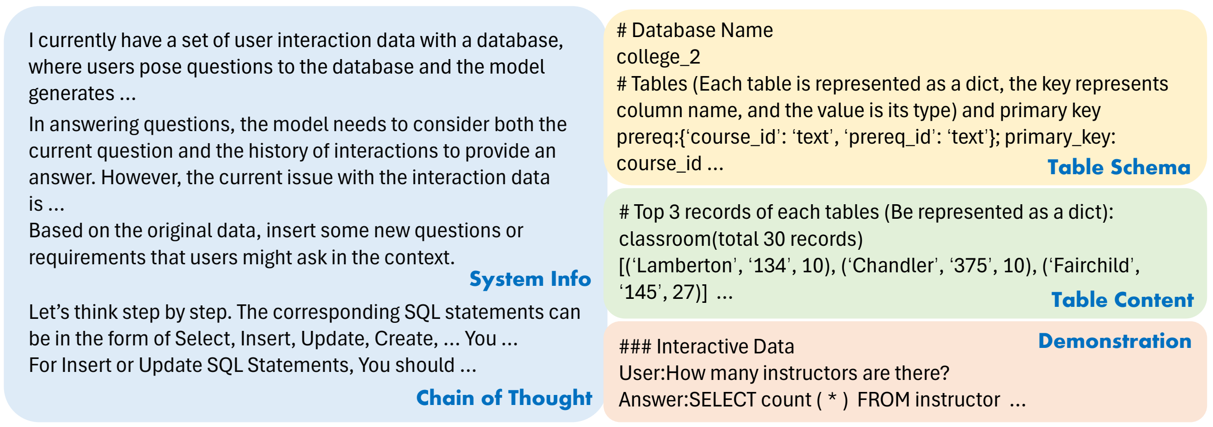
Our contributions are summarized as follows:

1. We present MultiSQL, a Multi-type, schema-integrated, and context-dependent Text2SQL dataset. By incorporating different types of SQL operations and integrating database schema dependencies in a long dialogue, we make our dataset a more comprehensive reflection of the complexity and dynamism in real-world database interactions.

2. In response to the challenges presented by our

Dataset	Cross Domain	Context Dependent	Schema Dependent	Multi SQL Types
Spider	✔	✘	✘	✘
Bird	✔	✘	✘	✘
CoSQL	✔	✔	✘	✘
SParC	✔	✔	✘	✘
Ours	✔	✔	✔	✔

Table 1: Comparison of various Text2SQL datasets, evaluating them against attributes such as cross-domain versatility, context dependency for accurate SQL generation, schema dependency for query formation, and the inclusion of multiple SQL types to assess the complexity of queries covered by each dataset.



I currently have a set of user interaction data with a database, where users pose questions to the database and the model generates ...
In answering questions, the model needs to consider both the current question and the history of interactions to provide an answer. However, the current issue with the interaction data is ...
Based on the original data, insert some new questions or requirements that users might ask in the context.

Let's think step by step. The corresponding SQL statements can be in the form of Select, Insert, Update, Create, ... You ...
For Insert or Update SQL Statements, You should ...

# Database Name
college_2
# Tables (Each table is represented as a dict, the key represents column name, and the value is its type) and primary key
prereq:{'course_id': 'text', 'prereq_id': 'text'}; primary_key: course_id ...

# Top 3 records of each tables (Be represented as a dict):
classroom(total 30 records)
[('Lamberton', '134', 10), ('Chandler', '375', 10), ('Fairchild', '145', 27)] ...

### Interactive Data
User:How many instructors are there?
Answer:SELECT count (*) FROM instructor ...

Figure 2: Structured prompt design for MultiSQL dataset construction, detailing components such as system information, table schema, and content, along with example interactions and chain of thought design.

dataset, we have accordingly developed methods and introduced new evaluation metrics.

3. Our experiments on models like GPT-3.5, GPT-4, and LLaMA2-7B effectively show the challenges posed by the MultiSQL dataset and the efficacy of our methods.

2 Related Work

Text-to-SQL aims to automatically translate natural language questions into SQL queries. The development of text-to-SQL datasets has evolved to more closely resemble real-world scenarios, reflecting an increase in complexity and a move towards more accurately simulating real-world data interactions.

Initially, datasets in this field were relatively simple and focused on single-domain scenarios. Early datasets such as GeoQuery (Zelle and Mooney, 1996), ATIS (Price, 1990), and Restaurant (Dahl et al., 1994)targeted specific information retrieval tasks within a limited domain. These datasets laid the groundwork for future advancements but were limited in their scope and complexity.

Considering that Text2SQL in real scenarios is an open-domain problem, the field has seen a shift towards cross-domain datasets, exemplified by WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018). These datasets broadened the scope from single-domain to cross-domain, requiring models to generalize across various domains. However, a common limitation in many cross-domain datasets is their focus on the database schema without adequately considering the specific values within the tables, diverging from the complexity of real-world databases.

To address this, datasets like KaggleDBQA (Lee et al., 2021), EHRSQL (Lee et al., 2022), SEDE (Hazoom et al., 2021), and MIMIC-SQL (Wang et al., 2020) were introduced. These datasets focused on large-value databases and professional SQL queries, thereby moving closer to real-world database applications. Bird further advanced this trend by incorporating noisy data and external knowledge into its structure to accurately depict the complexity of data in real-world business scenarios, making it more challenging and representative of real-world data processing scenarios.

Another significant development in Text-to-SQL datasets is the recognition that it's more natural for Text2SQL applications to involve users dynamically interacting with databases through dialogue. Therefore, context-dependent Text2SQL has been developed. SparC simulates user-database interactive dialogues, requiring models to understand and respond to context-dependent queries. CoSQL, on the other hand, is designed for building cross-domain, general-purpose database querying dialogue systems. It challenges systems in SQL-grounded dialogue state tracking, response generation, and user dialogue act prediction, closely mimicking real-world dialogue scenarios.

3 Dataset Construction

Due to CoSQL being a representative example of a context-dependent Text2SQL dataset, we adopted its database in our work. Building upon this foundation, we constructed a Text2SQL interaction dataset to further explore and address the complexities of natural language interfaces to databases. Due to the high cost and extensive time requirements associated with manual data annotations, we use large language models as an aid in the dataset construction process like much recent research does (Akyürek et al., 2023; Chen et al., 2023; Zhang et al., 2023b). We designed a set of prompt frameworks and used GPT-4 to generate data. Subsequently, we manually corrected and modified the generated data, and iteratively checked and improved the quality of the dataset. Below, we will provide a detailed description of the entire dataset construction process.

3.1 Prompt Design

Our prompt design, depicted in Figure 2, integrates several key components: (1) **System Information**, providing a task overview, dataset attributes, and input-output formats; (2) **Table Schema**, detailing the database structure including table and field names, data types, and keys; (3) **Table Content**, presenting the initial entries of each table and the total count of entries; (4) **Demonstrations**, offering initial *Select* queries from CoSQL to lay the groundwork for the model's understanding of database queries; and (5) **Chain of Thought**, employing a reasoning method to guide the model in formulating queries with accurate logic.

3.2 Data Generation

Here, we employed GPT-4 to generate data based on the above-mentioned prompt framework. Adding more detail to the situation, we generated 817 sets of dialogues, comprising a total of 9317 pairs of query-SQL data. Afterward, by filtering out the duplicates, we ultimately obtained 783 sets of dialogues, with 8923 pairs of query-SQL data.

3.3 Manual Correction and Improvement

However, the generation process by GPT-4 can only serve as an auxiliary tool. The data produced by GPT-4 still has some issues. Therefore, we invested substantial human effort to refine the base data generated by GPT-4, aiming to correct existing issues and enhance the overall quality of the dataset. This involved several key steps:

- **Grammar Correction:** We validated and corrected grammatical errors in 12% of the generated SQL statements to ensure adherence to SQL syntax standards. This included resolving 6% of errors due to conflicts in insert statement numbering from inability to accurately retrieve entry counts in the database, 4% for non-compliance with SQL syntax rules, and 2% for inserting duplicate fields.

- **Semantic Correction:** We identified and rectified semantic misalignments in 4% of the data, such as mapping queries to the incorrect tables, failing to understand temporal information in questions, and inferring missing context from the queries inaccurately

- **Contextual Relevance Improvement:** We inserted 334 query-SQL interactions based on dialogue context to enhance the relevance of semantic association within the dialogue context and the dependency on table schema, addressing the complexities of real-world database interactions.

Building on the previously mentioned steps, our team dedicated 114 person-hours to the post-processing, correction, and improvement of the generated data. This substantial effort in grammar correction, semantic accuracy, and contextual relevance was instrumental in ensuring the high quality of the dataset.

4 Data statistics and analysis

Table 2 presents the statistical information of our dataset and compares it with existing datasets. It reveals that our dataset comprises a total of 9257 query turns, covering 166 databases. This scale of query turns is comparable to mainstream datasets.

	Ours	**Spider**	**Bird**	**CoSQL**	**SParC**
Database nums	166	200	95	200	200
Table/DB	5.23	5.1	7.3	5.1	5.1
Total query turns	9257	10181	12751	11039	11257
Average query turns	11.82	–	–	3.67	3.0

Table 2: Statistical overview of our dataset and comparison with existing datasets

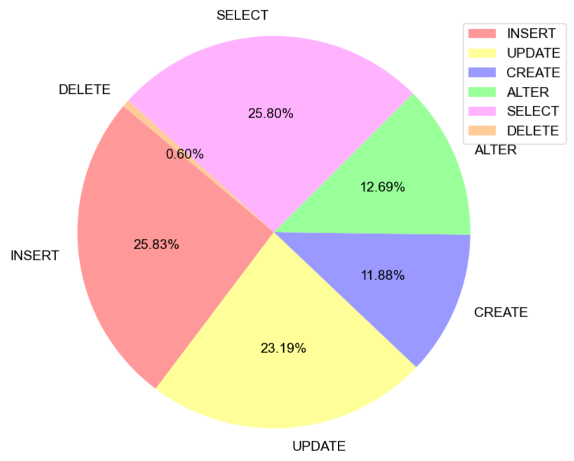


Figure 3: Distribution of SQL types in our dataset

Moreover, it's notable that our dataset has an average of 11.81 query turns per dialogue group, which is significantly higher than CoSQL and SParC.

Figure 3 shows the distribution of SQL types in our dataset, excluding *Delete* statements which are rare in practice. The balance among other operations is notable, with *Insert* at 25.83%, *Select* at 25.8%, and *UPDATE* at 23.19%. This reflects the varied use of SQL in real situations, enhancing our dataset's relevance and providing a broad testing ground for systems handling diverse SQL queries within dialogues.

5 Method

MultiSQL, with Diverse SQL Operations, Schema-Integrated Context, and Extended Dialogues, presents unique challenges to the method design.

First, the challenge arises from the need to understand the semantic context within dialogues and adapt to database schema changes. To address this, we integrate historical data, a strategy that involves incorporating past interactions into the model's current decision-making process. This approach ensures that the model not only grasps the immediate query but also contextualizes it within the dialogue's history, enhancing its ability to adapt to and reflect changes in the database structure accurately. By doing so, we mitigate issues related to semantic understanding and database adaptation, ensuring more accurate and contextually relevant predictions.

Second, the diversity in SQL types and the potential for error accumulation through extended dialogues require a robust mechanism to maintain model accuracy. Here, our solution is the implementation of a self-refinement mechanism. This process involves continuously analyzing model predictions for errors and refining the model's strategies based on feedback. Such a mechanism directly tackles the accumulation of errors by enabling the model to learn from its mistakes and adjust its approach for future queries, thus enhancing its reliability and accuracy in handling a wide range of SQL operations within prolonged dialogues.

Thus, we have devised a prompt framework that integrates historical data and incorporates a self-refinement mechanism. The framework contains several parts as shown in Figure 4:

- **System Initialization:** Starts with a prompt that outlines the task, database schema, and initial data context.
- **Integrating Dialogue History:** Adds previous dialogue excerpts to the current query for context.
- **Self-Refinement Mechanism:** Uses an SQL executor to validate and provide feedback on the model-generated SQL for correction and improvement.

In this way, our approach systematically addresses the challenges posed by MultiSQL through initial setup, context integration, and iterative refinement. Section 7 experiments further validate the effectiveness of our methodology.

6 Evaluation Metrics

In Text2SQL tasks, traditional evaluation metrics include *Exact Set Match* and *Execution Accuracy*.

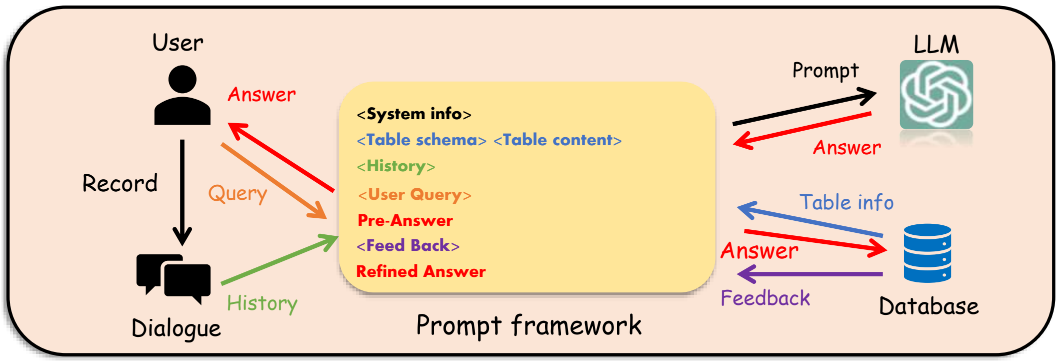


Figure 4: Our prompt framework for dynamic SQL query generation, including a user's queries and system's records, which in turn, synthesizes an answer from a database, with the ability for refined answers based on feedback loops

- **Exact Set Match:** This metric assesses the equivalence of the predicted query to the gold query across specific SQL components: *FROM*, *WHERE*, *GROUP BY*, etc. Each component is evaluated for an exact match between the predicted and gold queries. The predicted query is deemed correct if, and only if, all components match exactly with their counterparts in the gold query.

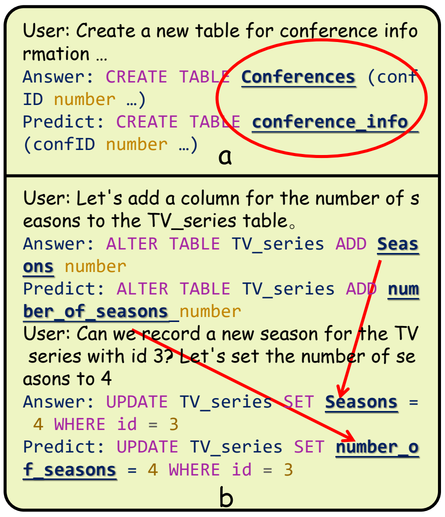
- **Execution Accuracy:** This metric executes both the predicted and gold SQL queries on the database and compares the result sets. If the result sets are identical, the queries are considered equivalent.

However, these metrics encounter specific challenges when applied to our dataset.

- **Challenges with Exact Set Match:** First, *Create* and *Alter* statements can lead to naming variations in fields, as depicted in Figure 5(a), where different but valid table names reflect the same query's intent. Second, as table structures change during dialogues (Figure 5(b)), accurately matching subsequent queries necessitates considering previous modifications

- **Challenges with Execution Accuracy:** Beyond *Select* statements, other types of SQL queries do not produce a direct return value, making it challenging to evaluate their execution effectiveness.

Consequently, to address the distinctive challenges of our dataset, we introduce two novel evaluation metrics: *Context-aware Match* which gauges


```
User: Create a new table for conference info
rmation …
Answer: CREATE TABLE Conferences (conf
ID number …)
Predict: CREATE TABLE conference_info
(confID number …)
```

```
User: Let's add a column for the number of s
easons to the TV_series table。
Answer: ALTER TABLE TV_series ADD Seas
ons number
Predict: ALTER TABLE TV_series ADD num
ber_of_seasons number
User: Can we record a new season for the TV
 series with id 3? Let's set the number of se
asons to 4
Answer: UPDATE TV_series SET Seasons =
 4 WHERE id = 3
Predict: UPDATE TV_series SET number_o
f_seasons = 4 WHERE id = 3
```

Figure 5: Evaluation challenges of MultiSQL: (a) The red circle indicates different table names can reflect the same query intent. (b) The red arrow shows changes in table structure across dialogue

the alignment of predicted and actual table structures within dialogue contexts, and *Database State Match*, which measures SQL effectiveness by examining database state alterations. Further details are provided in the following subsections.

6.1 Context-aware Match

To tackle the issue of field naming discrepancies, we first introduce fuzzy matching which aims to accommodate variations in naming conventions by allowing for a more flexible comparison between predicted and actual field names. Fuzzy matching operates under the principle that a match is recognized if:

1. The field name in the answer (s) is contained within the predicted field name (c), or vice versa;

2. Parts of s split by underscores (_) are found within c, or parts of c split by underscores are found within s.

Additionally, for operations like *Create* and *Alter* that change table structures, we track structural modifications by recording added (T_{add} and C_{add}), deleted (T_{del} and C_{del}), and altered (C_{alt}) elements. This leads to the creation of a mapping dictionary (M) that correlates predicted and actual database elements. During SQL evaluation, a context-aware match score of 1 is awarded for exact field matches as per M_{column}, with any deviation resulting in a 0.

$$\text{Score} = \begin{cases} 1 & \text{if named fields match according to } M_{column} \text{ and all non-named fields align perfectly} \\ 0 & \text{otherwise} \end{cases}$$

6.2 Database State Match

For evaluating non-*Select* statements, we refine our approach with the *Database State Match* metric. This metric contrasts the database states after executing the predicted SQL statement and the reference SQL statement, aiming to verify if these resulting states are identical. This comparison is essential for operations such as *Create*, *Alter*, and *Delete*, which impact the database's structure, and for *Insert* and *Update*, which alter its content.

The matching score K is thus recalculated to reflect the alignment between the predicted and actual outcomes on the database. The formula is adjusted as follows:

$$K = \begin{cases} \text{Structural Match}(D_{\text{pred}}, D_{\text{gold}}), \\ \qquad \text{for } \textit{Create}, \textit{Alter}, \textit{Delete} \\ \text{Content Match}(D_{\text{pred}}, D_{\text{gold}}), \\ \qquad \text{for } \textit{Insert}, \textit{Update} \end{cases}$$

In this context, D_{pred} and D_{gold} represent the database states after executing the predicted and gold standard SQL statements respectively. The Structural Match verifies the structural integrity and schema modifications aligned between D_{pred} and D_{gold}. while the Content Match ensures an exact content match in the database following the execution of both predicted and gold SQL statements.

7 Experiments

7.1 Experimental Setup

In our experiments, we employed GPT-3.5 (Ouyang et al., 2022), GPT-4 (Achiam et al., 2023), and LLAMA2-7B (Touvron et al., 2023). Additionally, we introduced domain-specific models for the code domain: TableLlama-7B (Zhang et al., 2023a), CodeLlama-7B (Roziere et al., 2023), and Deepseeker-Code-7B (Guo et al., 2024). We used a prompt framework and in-context learning to interact with these Large Language Models, maximizing their response capabilities to structured prompts. Specifically, for LLAMA2-7B, we adopted two approaches: a zero-shot setting where prompts were directly inputted for inference, and an instruction-tuned setting, where we utilized 662 groups from our dataset to construct 8006 instruction-tuning pairs. The remaining 121 groups were then employed for testing. For more implementation details, see Appendix A.

For prompt configurations, we experimented with several settings: (1) **Baseline**, which processes text directly to generate SQL statements without additional context; (2) **History**, which enhances inputs by appending historical dialogue data before SQL generation; and (3) **Self Refine**, which introduces a feedback loop from executing generated SQL to refine subsequent outputs.

Our main evaluation metrics are as follows: (1) **Execution Accuracy**, which focuses on the precision of executing SQL select statements; (2) **Context-aware Match**, adopting the previously mentioned strategy and assessing accuracy across all types of statements; and (3) **Database State Match**, evaluating the congruence of the database state post-execution, applicable to all statement types except select statements.

7.2 Experimental Results

Table 3 offers insightful results regarding the performance of different models and methods across key metrics such as *Context-aware Match*, *Execution Accuracy*, and *Database State Match*. In comparing models GPT-3.5, GPT-4, and LLAMA2-7B, the Self Refine method generally outperforms the Baseline and History methods across the board. For *Context-aware Match*, Self Refine achieves top scores in the *Create* and *Update* categories, with GPT-4 reaching 0.345 and 0.786 respectively. The *Execution Accuracy* for *Select* is also highest with Self Refine, scoring 0.701 for GPT-4 and 0.653 for GPT-3.5. *Database State Match* scores indicate Self Refine leads in *Insert*, *Create*, and *Update* actions, with GPT-4 scoring 0.586, 0.529, and 0.735 respectively.

Table 3: Comparative analysis of SQL generation performance among different models: GPT-3.5, GPT-4, and LLAMA2-7B, with a focus on context-aware matching, execution accuracy, and database state match for various SQL operations. Results are segmented into three methodologies: Baseline, History, and Self Refine, highlighting the incremental performance improvements with each approach.

		Context-aware Match						**Execution Acc**	**Database State Match**				
Models	**Method**	Select	Insert	Create	Update	Alter	Delete	Select	Insert	Create	Update	Alter	Delete
GPT-3.5	Baseline	0.313	0.336	0.273	0.595	0.288	0.911	0.536	0.358	0.437	0.520	0.723	0.929
	History	0.402	0.432	0.289	0.704	0.288	0.911	0.639	0.506	0.430	0.654	0.740	**0.947**
	Self Refine	0.408	0.427	0.287	0.728	**0.316**	**0.926**	0.653	0.514	0.428	0.673	0.747	0.946
GPT-4	Baseline	**0.496**	0.326	0.307	0.438	0.288	0.875	0.385	0.353	0.498	0.438	0.708	0.750
	History	0.497	0.423	0.313	0.704	0.292	0.911	0.600	0.522	0.479	0.642	0.695	0.911
	Self Refine	0.475	**0.469**	**0.345**	**0.786**	0.308	0.911	**0.701**	**0.586**	**0.529**	**0.735**	**0.780**	0.929
LLAMA2-7B	Baseline	0.000	0.020	0.013	0.035	0.039	0.000	0.000	0.012	0.040	0.011	0.070	0.000
	History	0.000	0.006	0.003	0.007	0.012	0.000	0.000	0.002	0.000	0.006	0.005	0.000
	Self Refine	0.000	0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LLAMA2-7B(tuned)	Baseline	0.036	0.023	0.011	0.176	0.096	0.000	0.025	0.023	0.018	0.088	0.208	0.250
	History	0.029	0.068	0.042	0.268	0.115	0.125	0.030	0.072	0.079	0.150	0.152	0.000
	Self Refine	0.059	0.074	0.039	0.155	0.107	0.000	0.025	0.077	0.079	0.111	0.219	0.125

Table 4: Context-aware Match performance of different methods across TableLlama-7B, CodeLlama-7B, and Deepseeker-Code-7B

		Context-aware Match					
Models	**Method**	**Select**	**Insert**	**Create**	**Update**	**Alter**	**Delete**
CodeLLAMA-7B	Baseline	0.027	0.052	0.017	0.130	0.173	0.250
	History	0.032	0.024	0.020	0.041	0.081	0.200
	Self Refine	0.020	0.009	0.000	0.003	0.000	0.000
TableLLAMA-7B	Baseline	**0.333**	0.051	0.006	0.029	0.051	0.250
	History	0.000	0.000	0.003	0.000	0.000	0.000
	Self Refine	0.000	0.000	0.000	0.000	0.000	0.000
DeepSeek-Coder-7B	Baseline	0.000	0.004	0.003	0.004	0.000	0.000
	History	0.095	0.062	**0.032**	0.235	0.170	0.344
	Self Refine	0.302	**0.353**	0.018	**0.622**	**0.335**	**0.625**

Table 5: Select Exec. Acc & Database State Match performance of different methods across TableLlama-7B, CodeLlama-7B, and Deepseeker-Code-7B.

		Exec. Acc	**Database State Match**				
Models	**Method**	**Select**	**Insert**	**Create**	**Update**	**Alter**	**Delete**
CodeLLAMA-7B	Baseline	0.064	0.101	0.072	0.102	0.257	0.500
	History	0.037	0.073	0.051	0.055	0.128	0.400
	Self Refine	0.006	0.006	0.000	0.003	0.000	0.000
TableLLAMA-7B	Baseline	0.010	0.049	0.012	0.021	0.062	0.375
	History	0.000	0.023	0.000	0.018	0.017	0.000
	Self Refine	0.000	0.023	0.000	0.018	0.017	0.000
DeepSeek-Coder-7B	Baseline	**0.457**	0.402	0.326	0.600	**0.750**	**0.750**
	History	0.322	**0.563**	**0.369**	**0.642**	0.697	0.625
	Self Refine	0.429	0.466	0.179	0.541	0.588	0.625

LLAMA2-7B, even when tuned, shows a stark contrast in performance compared to GPT models. The tuned LLAMA2-7B's best *Context-aware Match* scores after Self Refine are 0.059 for *Select* and 0.155 for *Update*, a considerable gap from GPT-4's performance. This highlights the LLAMA2-7B's limitations in complex SQL tasks and underscores the challenging nature of the dataset which demands robust contextual understanding and adaptability from models. The clear disparity in the results illustrates the importance of methodological refinement in achieving high accuracy on this demanding dataset.

Table 4 and Table 5 offer insightful results regarding the performance of different models and methods across key metrics such as *Context-aware Match*, *Execution Accuracy*, and *Database State Match*. These tables compare the models TableLlama-7B, CodeLlama-7B, and Deepseeker-Code-7B using Baseline, History, and Self Refine methodologies.

We can find that Deepseek-Coder-7B consistently performs the best, demonstrating its robust capability in handling various types of Text2SQL tasks. In contrast, CodeLLAMA-7B and TableLLAMA-7B struggle to match this performance, highlighting the challenging nature of our dataset. For the less capable models, CodeLLAMA-7B and TableLLAMA-7B, sophisticated prompt methods like History and Self Refine do not always yield the best results. This may reflect their weaker instruction comprehension capabilities. However, for Deepseek-Coder-7B, and the previously discussed GPT-3.5 and GPT-4, the benefits of the Self Refine method become increasingly apparent as their instruction comprehension abilities improve. This trend indicates that the performance of these models can be significantly enhanced with better contextual understanding and methodical refinement.

7.3 Detailed Analysis

We provide a detailed assessment of the *Create* and *Insert* SQL statements for models GPT-3.5, GPT-4, and LLAMA2-7B, with a focus on *Table Name Accuracy*, which is the correct prediction rate of table names, and *Field Match Ratio*, which is the precision of field name prediction. These detailed metrics are designed to finely gauge the models' predictive capabilities regarding database structure.

The experimental outcomes for GPT-3.5 and

Table 6: Detailed analysis of the *Create* and *Insert* SQL statements across GPT-3.5, GPT-4, and LLAMA2-7B. The *Table Name Accuracy* is the correct prediction rate of table names , and *Field Match Ratio* is the accurate ratio of field names in the *Create* and *Insert* statements.

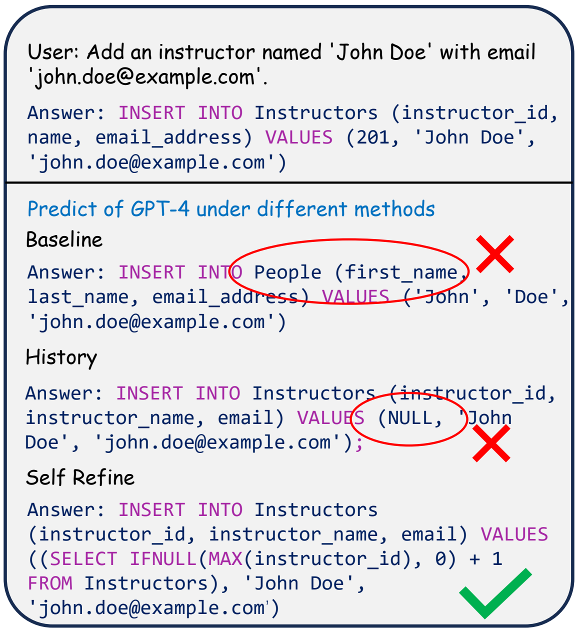
		Table Name Acc.		**Field Match Ratio**	
Models	**Method**	Create	Insert	Create	Insert
GPT-3.5	Baseline	0.975	0.954	0.722	0.930
	History	0.973	0.971	0.727	0.947
	Self Refine	0.968	0.956	0.728	0.933
GPT-4	Baseline	0.915	0.731	0.701	0.724
	History	0.891	0.893	0.693	0.873
	Self Refine	**0.984**	**0.988**	**0.769**	**0.963**
LLAMA2-7B	Baseline	0.191	<u>0.464</u>	0.070	<u>0.458</u>
	History	0.129	0.270	0.020	0.268
	Self Refine	0.027	0.146	0.012	0.146
LLAMA2-7B (tuned)	Baseline	0.075	0.199	0.032	0.194
	History	<u>0.207</u>	0.416	0.115	0.409
	Self Refine	0.185	0.239	<u>0.126</u>	0.236

GPT-4 demonstrate their proficiency. GPT-4 with the Self Refine method reaches near-perfect *Table Name Accuracy* scores of 0.984 for *Create* and 0.988 for *Insert*, and *Field Match Ratio* scores of 0.769 for *Create* and 0.963 for *Insert*. These findings robustly validate the effectiveness of the Self Refine, especially in predicting table names, where it exhibits an almost flawless performance.

In contrast, LLAMA2-7B's performance paints an interesting picture. The model, without any fine-tuning, records substantial *Field Match Ratio* scores of 0.464 for *Table Name Accuracy* in *Create* and 0.458 in *Insert*. These figures stand out against the backdrop of nearly zero scores in the primary experimental metrics of *Context-aware Match* and *Database State Match*. This indicates that LLAMA2-7B possesses a partial ability to predict *Insert* statements and shows a preference for generating table names and some database fields accurately. However, after fine-tuning, there is a noticeable decrease in performance, possibly because such tuning leads the model to approach *Insert* statement prediction from a more global perspective. This change might undermine its earlier predictive abilities, pointing to a nuanced trade-off between general and detailed SQL skills influenced by fine-tuning.

7.4 Case Study

In our case study on GPT-4 in Figure 6, we compared Baseline, History, and Self Refine methods using the same query. The Baseline method fails to deduce the correct table and column names due to lack of context. The History method corrects the table name but uses *NULL* for the ID, violating the non-null primary key constraint. The Self Refine



User: Add an instructor named 'John Doe' with email 'john.doe@example.com'.

```
Answer: INSERT INTO Instructors (instructor_id,
name, email_address) VALUES (201, 'John Doe',
'john.doe@example.com')
```

Predict of GPT-4 under different methods

Baseline

```
Answer: INSERT INTO People (first_name,
last_name, email_address) VALUES ('John', 'Doe',
'john.doe@example.com')
```

History

```
Answer: INSERT INTO Instructors (instructor_id,
instructor_name, email) VALUES (NULL, 'John
Doe', 'john.doe@example.com');
```

Self Refine

```
Answer: INSERT INTO Instructors
(instructor_id, instructor_name, email) VALUES
((SELECT IFNULL(MAX(instructor_id), 0) + 1
FROM Instructors), 'John Doe',
'john.doe@example.com')
```

Figure 6: A case study shows SQL predictions by GPT-4 using Baseline, History, and Self Refine methods for the same query. Red circles mark errors: Baseline fails to identify the correct table, and History can't generate a valid primary key. The green checkmark shows Self Refine's success, accurately predicting the table name and dynamically calculating the next primary key.

method successfully rectifies the table name and uses a subquery to compute the next ID, generating the correct SQL statement. This study highlights that the History method improves context perception over Baseline, while Self Refine excels by using database feedback to correct errors. Incorporating such feedback is crucial for generating contextually accurate and constraint-respecting SQL queries.

8 Conclusion

In conclusion, we introduce MultiSQL, a Multi-type, schema-integrated, and context-dependent Text2SQL dataset, designed to closely mirror the complexities and dynamism of real-world database interactions. By incorporating a diverse range of SQL operations and embedding database schema dependencies within extended dialogue interactions, MultiSQL offers a significantly more nuanced and challenging environment for Text2SQL applications.

Limitations

The MultiSQL dataset, while advancing the Text2SQL domain, encounters limitations in fully replicating the complexity of real-world database scenarios, potentially affecting its generalizability. With 166 databases, the variety, although extensive, may not encompass the vast diversity of real-world database schemas, limiting the dataset's applicability across different domains. Additionally, the refined evaluation metrics, though improved, might not capture all aspects of SQL query quality such as runtime efficiency and adherence to SQL writing best practices. This could lead to a gap in measuring the true effectiveness of SQL queries generated from natural language interactions, highlighting areas for future enhancement to bridge the gap between simulated environments and real-world database usage.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments. This work is supported by the National Natural Science Foundation of China (No. 62206126, 61936012 and 61976114).

References

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

Afra Akyürek, Eric Pan, Garry Kuwanto, and Derry Wijaya. 2023. DUnE: Dataset for unified editing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1847–1861, Singapore. Association for Computational Linguistics.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. TheoremQA: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7889–7901, Singapore. Association for Computational Linguistics.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

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

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

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 77–87, Online. Association for Computational Linguistics.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, pages 1–32.

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)*, pages 2261–2273, Online. Association for Computational Linguistics.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. 2022. Ehrsql: A practical text-to-sql benchmark for electronic health records. In *Advances in Neural Information Processing Systems*, volume 35, pages 15589–15601. Curran Associates, Inc.

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 Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll 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 F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, et al. 2023. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

P. J. Price. 1990. Evaluation of spoken language systems: the ATIS domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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*, pages 350–361.

Tao Yu, Rui Zhang, Heyang 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 Lasecki, and Dragomir Radev. 2019a. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

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*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

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. 2019b. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023a. Tablellama: Towards open large generalist models for tables.

Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. 2023b. CRT-QA: A dataset of complex reasoning question answering over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2131–2153, Singapore. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Implementation Details

In our Text2SQL experiments, we implemented specific configurations for models in the GPT series. For GPT-3.5, we employed the GPT-3.5-turbo model, and for GPT-4, we used the GPT-4-1106-preview version. Both models were set with a temperature of 0 to ensure the determinism and stability of the generated results.

For fine-tuning LLAMA2-7B, we crafted a dataset comprising 783 groups. To prepare for instruction tuning, we utilized 662 of these groups to construct 8006 instruction-tuning pairs. This was done by segmenting user-answer pairs within dialogues. The remaining 121 groups were reserved for testing purposes. The fine-tuning process was carried out using the LoRA technique, with a LoRA rank of 8 and a LoRA alpha of 32. We set the batch size to 4 and the learning rate to 1e-4. The fine-tuning was conducted on an NVIDIA Tesla V100 32GB GPU. For the inference output from LLAMA2-7B, due to the model's limitations, the generated content contained some redundant information. To address this, we employed GPT-3.5 to extract the SQL statements from the generated content, which were then used as the predictive results of the model.