EVOSCHEMA: TOWARDS TEXT-TO-SQL ROBUSTNESS AGAINST SCHEMA EVOLUTION

Tianshu Zhang
The Ohio State University
Columbus, OH
zhang.11535@osu.edu

Kun Qian
Adobe Inc.
Seattle, WA
kunq@adobe.com

Siddhartha Sahai
Adobe Inc.
Seattle, WA
siddharthas@adobe.com

Yuan Tian
Purdue University
West Lafayette, IN
tian211@purdue.edu

Shaddy Garg
Adobe Inc.
Bangalore
shadgarg@adobe.com

Huan Sun
The Ohio State University
Columbus, OH
sun.397@osu.edu

Yunyao Li
Adobe Inc.
San Jose, CA
yunyaol@adobe.com

ABSTRACT

Neural text-to-SQL models, which translate natural language questions (NLQs) into SQL queries given a database schema, have achieved remarkable performance. However, database schemas frequently evolve to meet new requirements. Such schema evolution often leads to performance degradation for models trained on static schemas. Existing work either mainly focuses on simply paraphrasing some syntactic or semantic mappings among NLQ, DB and SQL, or lacks a comprehensive and controllable way to investigate the model robustness issue under the schema evolution, which is insufficient when facing the increasingly complex and rich database schema changes in reality, especially in the LLM era.

To address the challenges posed by schema evolution, we present EvoSchema, a comprehensive benchmark designed to assess and enhance the robustness of text-to-SQL systems under real-world schema changes. EvoSchema introduces a novel schema evolution taxonomy, encompassing ten perturbation types across column-level and table-level modifications, systematically simulating the dynamic nature of database schemas. Through EvoSchema, we conduct an in-depth evaluation spanning different open-source and closed-source LLMs, revealing that table-level perturbations have a significantly greater impact on model performance compared to column-level changes. Furthermore, EvoSchema inspires the development of more resilient text-to-SQL systems, in terms of both model training and database design. The models trained on EvoSchema's diverse schema designs can force the model to distinguish the schema difference for the same questions to avoid learning spurious patterns, which demonstrate remarkable robustness compared to those trained on unperturbed data on average. This benchmark offers valuable insights into model behavior and a path forward for designing systems capable of thriving in dynamic, real-world environments.

PVLDB Reference Format:
Tianshu Zhang, Kun Qian, Siddhartha Sahai, Yuan Tian, Shaddy Garg, Huan Sun, and Yunyao Li. EVOSCHEMA: TOWARDS TEXT-TO-SQL ROBUSTNESS AGAINST SCHEMA EVOLUTION. PVLDB, 18(10): 3655 - 3668, 2025.
doi:10.14778/3748191.3748222

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

1 INTRODUCTION

Text-to-SQL parsing aims to translate natural language questions (NLQs) into SQL queries given a database schema, enabling the development of natural language interfaces that allow users to query data and invoke services without requiring programming skills [18, 27, 29, 32, 33, 36]. Existing neural text-to-SQL models have achieved remarkable performance on existing benchmarks [18, 32], which play an important role in empowering different platforms such as business and marketing platforms [26, 34] and being integrated into virtual assistants to enable real-time data query and analysis [4].

However, database schemas are not static; they frequently evolve to accommodate new use cases and improve efficiency [3, 11]. For instance, depending on the scenario, a large patient table might be merged from or split into two tables: a patient information table and a patient diagnosis table (Figure 1-c), to reduce redundancy, enhance data integrity, and optimize performance [14]. Such schema evolution occurs frequently, which often leads to distribution shifts [13, 24] such as nomenclature shifts, data granularity shifts, table and column relation shifts and schema complexity shifts. These distribution shifts can cause significant performance degradation when the model trained on old database schema is adapting to new schema designs.

Proceedings of the VLDB Endowment, Vol. 18, No. 10 ISSN 2150-8097.
doi:10.14778/3748191.3748222

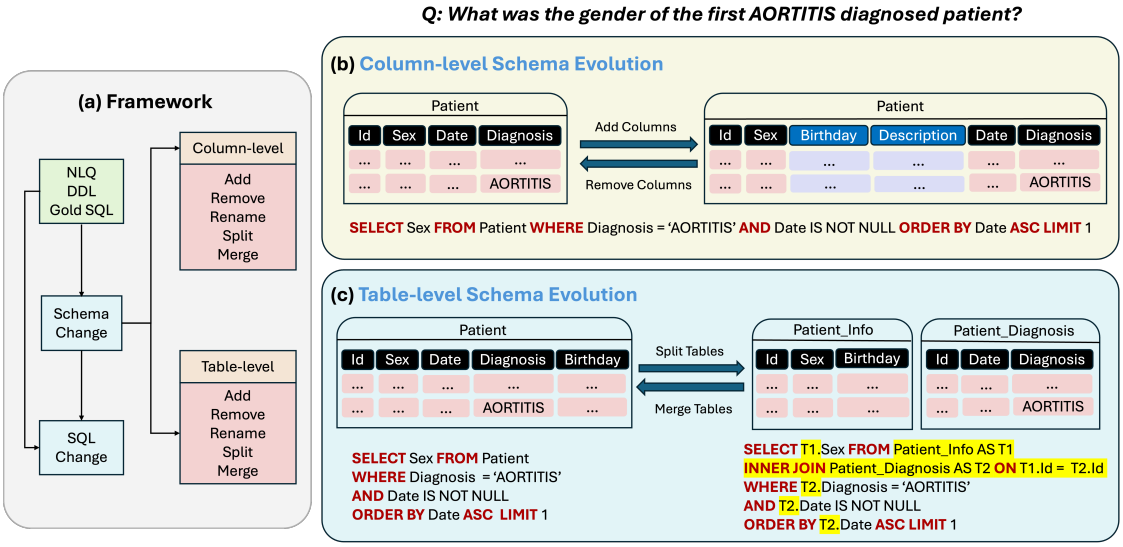

Figure 1: The left (a) is the overview of the framework to collect EvoSchema dataset. The top right (b) is a column-level schema evolution example; the bottom right (c) is a table-level schema evolution example.

This challenge highlights a critical issue in model robustness: how well can a text-to-SQL model adapt to changes in the database schema? Recent studies introduce evaluation benchmarks designed to expose robustness issues by perturbing NLQs, databases or SQL queries [2, 6, 20, 23]. However, these studies have at least one of the following limitations: 1) mainly focus on the syntactic paraphrasing or simple semantic mappings among NLQ, DB and SQL [2, 6]; (2) lack a taxonomy of comprehensive schema evolution types [23]; (3) only focus on schema evolution that does not lead to SQL changes [20]. These efforts are insufficient in the face of increasingly complex and rich database schema changes found in reality. Meanwhile, while it is natural to consider collecting new data after schema evolution for retraining a model, repeating the entire model training life cycle frequently can be costly in terms of both time and resources.

Under this background, we seek to answer the following two questions: (1) How sensitive are existing text-to-SQL models to various types of database schema changes? (2) How can we train a more robust text-to-SQL model that not only performs well on existing database schemas but also adapts effectively to schema changes? Towards this end, we introduce EvoSchema, a new dataset that covers a wide range of realistic schema design changes by perturbations on BIRD [18]. As illustrated in Figure 1 and Figure 2, EvoSchema builds upon our newly defined taxonomy, which encompasses a total of ten types of perturbations over schema, covering both column-level and table-level changes. Column-level perturbations include adding, removing, renaming, splitting and merging columns, while table-level perturbations involve adding, removing, renaming, splitting, and merging tables. We keep the NLQs fixed and examine the robustness of a model under different granularities of schema evolution, and show that existing models are more easily affected by table-level perturbations than column-level perturbations.

Moreover, the training set in EvoSchema can be used to enhance models' robustness. The models can be trained with the same questions but coupled with different schema designs to generate the corresponding SQL queries. This training procedure forces the model to distinguish the schema difference which can help models gain a stronger ability to recognize the correct table and column relation and map them to the questions. Our experimental results demonstrate that the perturbation training data in EvoSchema can help train better text-to-SQL models, which are more robust to different schema evolution types on average, especially on table-level perturbations.

In summary, our main contributions are as follows:

- We formulate a critical schema evolution adaptive text-to-SQL problem and present a new dataset, EvoSchema to study this problem. We introduce a comprehensive taxonomy of the schema evolution types and build the datasets based on the taxonomy to get realistic schema designs by column-level and table-level perturbations on BIRD.
- We conduct thorough and comprehensive assessment of model robustness against various schema perturbations spanning different open-source and closed-source LLMs on our evaluation benchmark, and find that table-level perturbations have a significantly greater impact on model performance compared to column-level changes. Besides, we introduce two evaluation metrics: Table Match F1 and Column Match F1, to rigorously evaluate the performance of text-to-SQL models under schema evolution scenarios and provide fine-grained insights into model robustness.
- Our constructed training set inspires a new training paradigm: augmenting the existing training data with different schema designs, which not only increase the data diversity, but also force the model to distinguish the schema

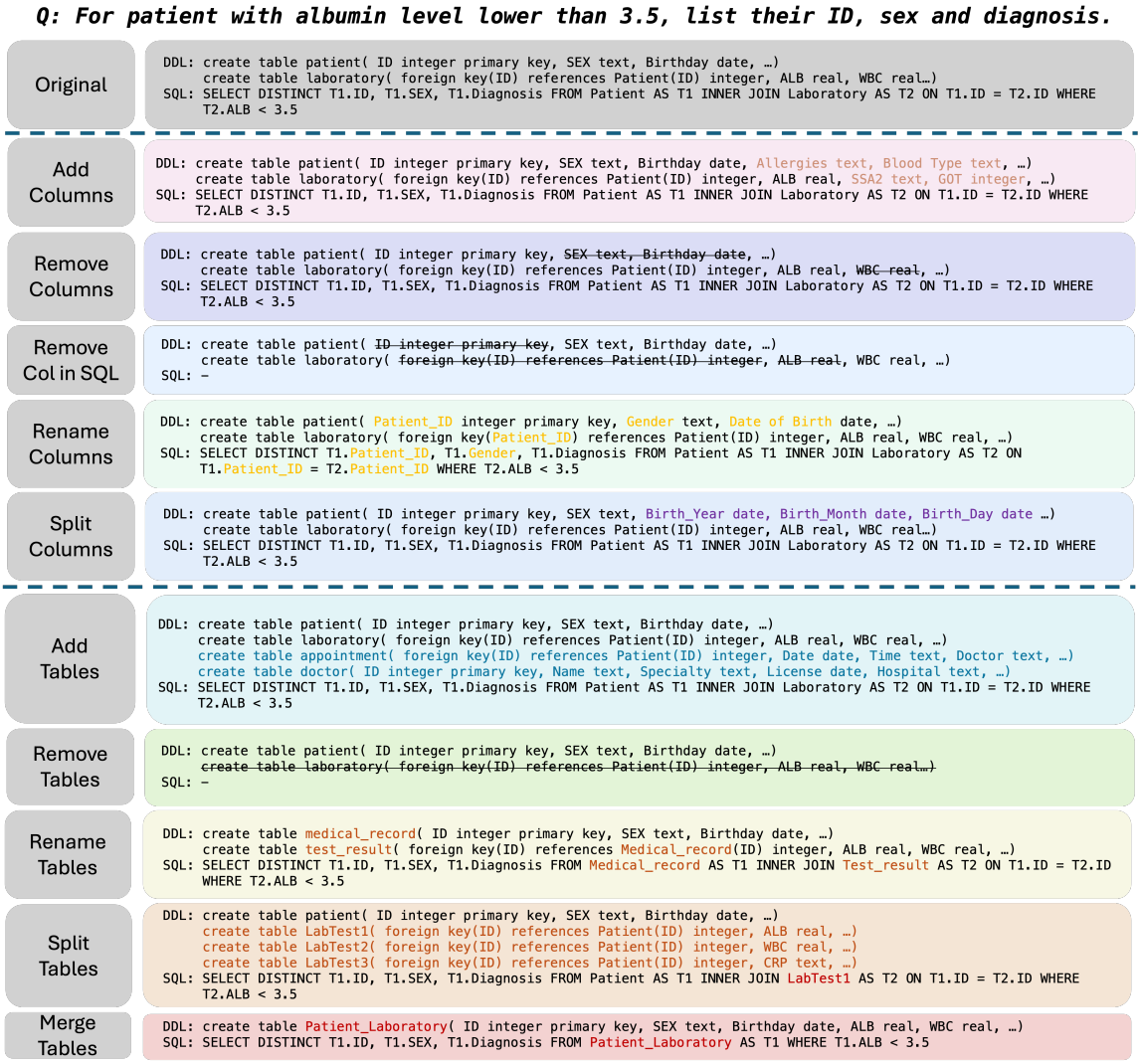

Figure 2: An overview of different perturbation types of EvoSchema. The top is an unperturbed example in BIRD [18]; the middle is the column-level perturbation; the bottom is the table-level perturbation. "Remove Col in SQL": remove columns that appear in gold SQL; "Remove Tables": the relevant tables appear in gold SQL are removed. Thus there is no gold SQL for these two cases. Note we don't illustrate "Merge Columns" in the figure as this example is not suitable for applying merging column changes.

difference during training. Our approach yields better text-to-SQL models that achieve up to 33 points gain on different types of schema perturbation evaluation data, compared to models trained on unperturbed, original training data.

2 RELATED WORK

Robustness in Text-to-SQL: Existing research on text-to-SQL robustness is mainly two-fold: robustness evaluation and robustness training. Recent studies introduce evaluation benchmarks designed to expose robustness issues by perturbing NLQs, databases or SQL queries. However, these studies tend to focus on syntactic paraphrasing or simple semantic mappings, such as different representations of numbers or name abbreviations across NLQ, DB, and SQL [2, 6]. While some work analyzes schema changes, they mainly focus on irrelevant column modifications that do not affect SQL [20] or with limited perturbation types [23]. These efforts are insufficient in the face of increasingly complex and rich database schemas found in modern datasets. Though FootballDB [8] tackles a similar schema design problem for better SQL written, they focus on reducing multiple foreign key mappings among tables and reducing the JOIN paths in the SQL. Different from theirs, we tackle the schema evolution problem, which is not only for the schema design on the existing data, but also needs to consider how new data and information will change the schema design. Besides, we

approach it through a different angle, where our scheme design contains 10 column-level and table-level changes. And our provided schema evolution framework allows us to try different schema design on multiple databases to get more generalizable findings, while FootballDB [8] can only support the exploration on a single database. Moreover, the advent of LLMs has mitigated many linguistic challenges, further emphasizing the need for robust adaptation to structural changes in database schemas. For robust training, existing methods employ strategies like decomposing tasks so that models generate each sub-clause individually before merging them [9], or using execution-guided decoding to eliminate incorrect subclauses [30]. While these approaches focus on enhancing various aspects of text-to-SQL robustness, our work specifically addresses the challenge of schema evolution.

LLMs for Text-to-SQL: Most recently, the LLM-based approaches for text-to-SQL are mainly two-fold: in-context learning [10, 15, 16, 27, 35] and finetuning [15–17, 38]. The former prompts proprietary LLMs such as GPT series [1] and Claude [2] for SQL generation without additional model training, while the latter involves adapting open-source LLMs to text-to-SQL datasets, tailoring these models directly to the task through supervised learning. These models are designed for question understanding, schema comprehension and SQL generation, which have achieved remarkable performance on the existing open benchmarks [18, 32]. Liu et al. [19] provides a comprehensive review of the NL2SQL lifecycle, covering models, benchmarks, data synthesis, evaluation, and error analysis. While it identifies schema variation as a challenge, it does not explore it in depth. Our work focuses specifically on schema evolution robustness by evaluating recent and powerful LLMs (e.g., Code Llama, Mistral, SQLCoder, LLaMA 3, GPT-3.5, GPT-4) without preprocessing or postprocessing. We introduce `EvoSchema`, a benchmark with controlled schema perturbations that guides both evaluation and structured training data synthesis. In addition to standard execution accuracy and human evaluation, we propose two fine-grained metrics: Table Match F1 and Column Match F1 that directly reflect our table-level and column-level perturbation taxonomy. Li et al. [15] evaluates LLMs on unperturbed Spider and BIRD datasets and also experiments on natural language variation but keep schema and SQL fixed; in contrast, our work systematically varies the schema while keeping the natural language fixed.

3 EVOSCHEMA DATASET

3.1 Background

In the dynamic landscape of databases, schemas frequently evolve to meet new demands, introducing significant challenges for text-to-SQL models [3, 5]. These schema changes can vary widely, from minor modifications to complete restructuring, and can significantly impact the performance of models trained on static schemas. In realistic scenarios, a database can often contain a large number of tables, and only several related tables are responsible for a natural language question (NLQ). In our experiment, we represent the relevant database schema using Data Definition Language (DDL) [3] and combine it with the NLQ as input. This input is then used to prompt the model to generate the corresponding SQL query.

[1] https://platform.openai.com/docs/models

[2] https://www.anthropic.com/news/claude-3-family

[3] DDL defines the structure and properties of a database, providing detailed information necessary for database creation, including column types and primary/foreign keys.

3.2 Rationale for Schema Evolution Types

When a database schema evolves, it can induce distribution shifts in the data that may impact model performance. We categorize potential distribution shifts into four types: nomenclature shifts, data granularity shifts, table and column relation shifts, and schema complexity shifts. (1) Nomenclature shifts occur when tables and columns are renamed, which may alter the convention of the established terminology within the schema. For example, tables originally named “Products", “Customers", and “Orders" might be renamed to “Items", “Clients", and “Purchases", respectively. Such changes often reflect updates in business terminology or compliance with new standards. A desired model should handle those nomenclature shifts to adapt to the new terminology. (2) Data granularity shifts arise from adding or removing columns or tables, which changes the level of detailedness captured in the database. For instance, an “Employee" table with a single “ContactNumber" field might involve another two separate “WorkContact" and “PersonalContact" fields later. This increases the data granularity to meet new requirements, necessitating models to adapt to more complex and detailed semantics. (3) Table and column relation shifts and schema complexity shifts mainly result from restructuring tables through splitting or merging. This process can highly affect how each table is related to other tables by which column. Both the primary keys and foreign keys may change along with the table restructure. Besides, the schema complexity may change when multiple tables merge from or split into one table. A desired model is expected to be robust to such changes. By categorizing the distribution shifts caused by schema evolution, we can more effectively understand and evaluate a model’s capacity to adapt to changes in the underlying database schema.

3.3 Schema Evolution Synthesis Framework

Our study aims to cover comprehensive potential schema evolution types, which can foster the robustness evaluation of the existing text-to-SQL models and inspire robust model training. We synthesize all the schema evolution types through hybrid strategies, which will leverage both the heuristic rules to guarantee the data quality and LLMs to ensure diversity.

Broad Coverage of Different Schema Evolution Types: We aim to encapsulate a broad range of schema evolution types, recognizing their prevalence and impact in real-world scenarios. Specifically, our schema evolution taxonomy includes both column-level and table-level perturbations, which are categorized into ten distinct types. Column-level perturbations comprise five types: adding, removing, renaming, splitting and merging columns, where modifications are restricted to the columns within existing tables. Table-level perturbations encompass five types: adding, removing, renaming, splitting, and merging tables. These perturbations occur frequently in practice, underscoring the need for text-to-SQL models that can robustly handle such changes.

Hybrid Data Synthesis Strategies: To ensure both diversity and quality in the generation of schema perturbations, we employ a combination of heuristics and GPT models to synthesize various

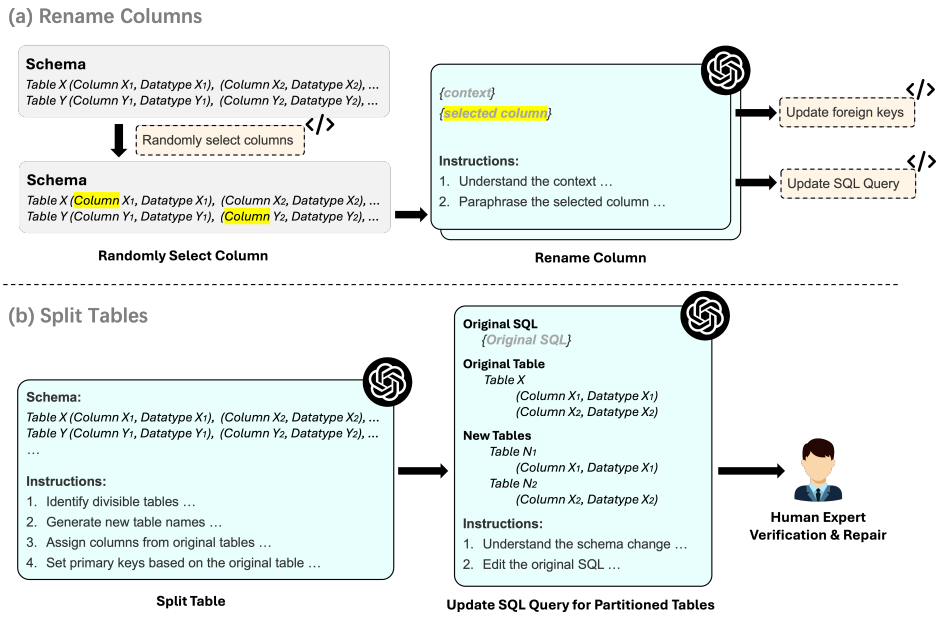

Figure 3: This figure shows two examples of our data collection procedure of EvoSchema. The top (a) is a "rename columns" data collection procedure; the bottom (b) is a "split tables" data collection procedure. The blue box indicates prompting GPT models for the generation. "</>" means programmatically processing the data.

perturbation types. For each given seed instance in BIRD [18], consisting of a *<NLQ, relevant schema, SQL>* triple, we maintain the natural language question (NLQ) fixed across all perturbation types, while only modifying the relevant schema. The corresponding SQL query is adjusted as necessary to remain consistent with the changes in the database schema.

3.4 Seed Dataset Selection

For building Evoschema benchmark, we utilize the BIRD [18] dataset as the seed data, which is specifically designed for the text-to-SQL task. Compared to Spider [32], which is commonly used to study text-to-SQL robustness, BIRD features more intricate, realistic, and extensive databases, as well as more complex SQL queries that include keywords often missing in Spider. BIRD consists of NLQs, corresponding database schemas, and gold SQL queries and encompasses a wide range of real-world database scenarios, which provides a robust foundation for evaluating the performance of models in translating NLQs into SQLs.

Schema Perturbations: To evaluate the robustness of the text-to-SQL models, EvoSchema not only includes the BIRD dataset in their original form but also augmented it with various column-level and table-level schema perturbations. We ensure that the NLQs remain fixed, while the schema and SQL queries are adjusted as necessary to reflect the changes introduced by our perturbations. We follow the standard train/dev split provided with BIRD, and apply all the perturbations on both training data and evaluation data. The data statistics of EvoSchema are in Table 2 and the examples of different perturbation types are in Figure 2.

3.5 Data Generation

We design a framework to simulate different types of schema perturbations in a configurable way. For adding or renaming columns, both the modified column size and the column position in the tables are set randomly, and we set the original column size in the table as the maximum number of columns to be changed. For removing columns, we can randomly remove important or unimportant columns from the existing relevant tables. The important columns are the columns that appear in the gold SQL, which will inevitably affect the prediction. For adding, removing, or renaming tables, we randomly add, remove or rename one or multiple tables.

Schema Change: To ensure the diversity and reasonability of the synthesized schema, we leverage the capabilities of GPT-3.5 and GPT-4 to synthesize realistic and contextually appropriate columns or tables, which help effectively produce high-quality synthetic data that meets our requirements. For adding or renaming columns and tables, we input the existing relevant tables to GPT-3.5, and let the model generate the potential tables or columns that fit the context. For splitting tables or merging tables, since they are more complex than other perturbations, we use GPT-4 to choose the tables that can be split or merged and then use the modified tables to replace the original ones. For adding or renaming columns and tables, we apply heuristics to filter out the repeated ones in the synthesized tables or columns. Besides, to ensure the correct relationship among different tables after modifying the schema, we apply heuristics to ensure all the foreign keys change along with their referenced table names and column names. When removing columns or tables, any foreign keys in other tables that reference the removed columns or tables will be removed as well.

SQL Change: To ensure the consistency of the *<NLQ, relevant schema, SQL>*, after we change the relevant table schema, we revise the gold SQL accordingly. Since the NLQs are the same for adding or removing columns and tables, and the schema evolution here doesn't affect answering the questions, we keep the gold SQL unchanged for these perturbation types. For renaming columns or tables, we revise gold SQL if they appear in the gold SQL. For table splitting or merging, due to the complexity and variation in the required SQL changes, we use GPT-4 to revise the gold SQL. This revision is based on the mappings from the original to the new tables and columns, as well as the necessary adjustments to the JOIN paths. We manually check the edited gold SQL for the evaluation benchmark to make sure they are correct.

3.6 Data Collection of Each Perturbation Type

We first define heuristics for different perturbation types, then combine both GPT models' generation ability and programming to collect the data. Finally, we incorporate a human verification stage to control the data quality. Here are some general heuristics we should consider to maintain consistency and avoid conflicts when manipulating data: 1) Preserve Meaning: For renaming, the new column or table name should reflect the same meaning as the original name to avoid semantic confusion. 2) Avoid Conflicts: Ensure that the new column or table name does not conflict with existing column or table names within the same or other tables in the database. 3) Update References: Update all references to the new column or tables in foreign keys in other tables. 4) Revise SQL: Update all SQL queries referencing the new columns or tables to work correctly after the renaming. These heuristics aim to ensure that those perturbations are performed systematically, maintaining the database's integrity and compatibility with SQL queries. The details for each perturbation type are as follows:

Add columns: we input both the table name and all of its column names and data types to GPT-3.5 and prompt it to generate multiple column names and their corresponding data types that are suitable and congenial with reason and common sense given the current scenario, and prompt GPT-3.5 don't generate the column names that have the similar meaning with the existing input column names. Then we add a heuristic guarantee to filter out the redundant columns if the generated column names are repeated. These synthesized columns are then randomly inserted into the relevant tables. Notably, both the NLQ and the gold SQL remain unchanged during this process.

Remove columns: We randomly eliminate columns from the given schema, ensuring that the removed columns do not appear in the gold SQL query. Again, the NLQ and the gold SQL are kept fixed during this operation.

Remove columns in gold SQL: In this scenario, we randomly remove columns from the schema, specifically targeting those referenced in the gold SQL query. As a result, the gold SQL becomes invalid. Instead, we use the response "The given column information is insufficient to generate an SQL query to answer the question" as the ground truth.

Rename columns: as Figure 3 (a) shows, we input both the table name and all of its column names and data types to GPT-3.5. We randomly select multiple column names and their data types and prompt GPT-3.5 to generate similar, context-appropriate names. These synthesized names replace the original column names. In addition, in order to maintain the correctness of the relationship among the tables, If the column in one table has been renamed, we will also rename the foreign keys in other tables if those columns reference the renamed one. We also revise gold SQL accordingly to ensure that the revised schema and gold SQL remain aligned with the unchanged NLQ.

Split columns: Since columns such as name, address, and date are often stored in more fine-grained formats in real-world databases (e.g., a full name split into first and last name; a date split into year, month and day; an address split into state, city and street, etc), we identify examples in BIRD dev set that involve these attributes and manually split the corresponding columns into finer-grained columns for evaluation. As these changes affect the structure of the gold SQL queries, we manually revise the gold SQL to reflect the updated schema. For the training set, we similarly select examples in BIRD train set involving name, address, or date, and use Claude 3.5 to synthesize the corresponding fine-grained columns and update the gold SQL accordingly.

Merge columns:As the reverse of column splitting, we simulate more abstract column representations commonly seen in real-world databases (e.g., combining first and last name into full name; year, month, and day into date; state, city, and street as address). We identify relevant examples in the BIRD dev set and manually merge fine-grained columns, updating the gold SQL accordingly. For training, we apply the same strategy to the BIRD train set and use Claude 3.5 to synthesize the merged schema and update the gold SQL.

Add tables: We randomly add irrelevant tables to each question, and these tables are still in the same database as the relevant tables in BIRD. The original BIRD datasets guarantee that no different tables in their database can lead to alternative correct SQL answers. The tables added don't affect the NLQ and the gold SQL.

Remove tables: In this scenario, we randomly remove tables from the relevant schema, which are referenced in the gold SQL query. As a result, the gold SQL becomes invalid. Instead, we use the response "The given table information is insufficient to generate an SQL query to answer the question" as the ground truth.

Rename tables: we input both the table name and all of its column names and data types to GPT-3.5. We randomly select one or multiple table names and prompt GPT-3.5 to generate similar, context-appropriate names. These synthesized names replace the original table names. In addition, in order to maintain the correctness of the relationship among the tables, we will also rename the foreign keys in other tables if they reference the renamed table. Finally, the table names in the gold SQL will also be renamed.

Split tables: as Figure 3 (b) shows, we input both the table name and all of its column names and data types to GPT-4. We prompt GPT-4 to identify tables that can be logically divided into two or more smaller tables. Using GPT-4, we generate new table names and distribute the columns of the original table among the new tables in a contextually appropriate manner. The primary key in the original table will be copied into all the new tables after splitting. The gold SQL is revised by GPT-4 to reference the newly created tables, ensuring consistency across all components. We also manually check the new gold SQL to make sure it's correct.

Table 1: Statistics of EvoSchema compared with existing benchmarks. "Tab": tables; "DB": database; "Col": columns; "PK": primary keys; "FK": foreign keys.

Perturbation Data	Column-level	Table-level	Schema Evolution Affects SQL	Multiple DB	Seed Data	Features of Seed Data (Average)				
						Tab/DB	Col/DB	Col/Tab	PK/DB	FK/DB
FootballDB [8]	-	reduce PK/FK references, reduce JOIN paths	✓	✗	FIFA World Cup [1]	15	107	7.1	-	16
Dr.Spider [2]	Rename	✗	✓	✓	Spider [2]	5.1	22.1	5.4	3.7	3.2
ADVETA [23]	Add; Rename	✗	✓	✓	Spider [2]	5.1	22.1	5.4	3.7	3.2
MT-TEQL [20]	Add; Remove; Shuffle; Rename	Split; Merge; Shuffle	✗	✓	Spider [2]	5.1	22.1	5.4	3.7	3.2
EvoSchema (Ours)	Add; Remove; Rename; Split; Merge	Split; Merge; Rename; Add; Remove	✓	✓	BIRD [18]	7.3	72.5	10.6	6.5	9.3

Merge Tables: We select two or more related tables and combine them into a single table. GPT-4 is used to generate a suitable name for the merged table, and the columns from the original tables are consolidated under this new table. More concretely, the GPT4 is prompted to 1) copy all the primary key columns of the original tables to the new tables after merging, but only keep one of them as the primary key of the new table, and make others as the regular columns. 2) if the primary key columns in these two original tables are the same, then just keep one in the new table after merging. 3) when merging tables, if there are two columns not the primary key column but with the same names in the original tables, revise their column names accordingly to make them different when merging them into the new table. Finally, the gold SQL is updated by GPT-4 accordingly. We also manually check the new gold SQL to make sure it's correct.

Quality Control: To ensure high-quality data in EvoSchema, we leverage advanced language models and rigorous human validation. Specifically, we use GPT-3.5 to generate synthesized column and table names and data types (only for columns) when adding or renaming are required. We randomly choose 200 generated examples to do manual review and reveal that GPT-3.5 demonstrates a strong understanding of the input context, effectively generating names that meet our requirement. For more complex operations, such as splitting or merging tables, we utilize the capabilities of more powerful GPT-4 to handle both schema changes and corresponding SQL modifications with high accuracy.

To complement these automated processes, we engaged five annotators with substantial SQL expertise to carefully review cases involving complex schema transformations. Annotators validated and, where necessary, manually corrected the generated gold SQL queries to ensure correctness and alignment with the modified schemas. To further enhance reliability, we implemented cross-validation by assigning complex cases to multiple annotators and resolving discrepancies through discussion or consensus. This combination of advanced AI tools and meticulous human review ensures that EvoSchema maintains a robust and accurate benchmark, faithfully reflecting real-world schema evolution scenarios.

Cost Analysis: We have 1.5K split-table examples and 1.1K merge-table examples requiring human verification. Among the split examples, 1.1K are relatively simple and take approximately 3 minutes each to verify, while the remaining 0.4K are more complex and require about 7 minutes each—totaling roughly 100 hours. For the merge-table examples, 0.8K are simple (3 minutes each) and 0.3K are complex (7 minutes each), amounting to approximately 75 hours. Note this manual effort was for curating the evaluation data, not the training data. Our training data is generated entirely automatically without any human annotation or manual verification. Our analysis also indicates that LLM-generated split and merge tables include around 30% low-quality data, underscoring the need for careful human validation for these two types.

3.7 Comparison with Existing Benchmarks

EvoSchema, as presented in Table 1, introduces a comprehensive and unique taxonomy for evaluating models' behavior under the impact of schema evolution on SQL queries, distinguishing itself from other benchmarks like Dr.Spider [2], ADVETA [23], MT-TEQL [20] and FootballDB [8]. Unlike Dr.Spider and ADVETA, which focus on limited perturbations such as column renaming and additions, EvoSchema encompasses a broader range of transformations, including adding, removing, renaming, splitting and merging at both the column level and table level. This diversity allows for testing systems under realistic and dynamic schema evolution scenarios. Furthermore, while MT-TEQL includes a variety of perturbations, it only modifies the columns not mentioned in the SQL which does not consider the impact of schema evolution on SQL directly. EvoSchema uniquely integrates schema evolution with its effects on SQL queries, enabling evaluation of models in environments that closely mimic real-world database evolution challenges. Different from FootballDB [8] which mainly restructures schema to reduce foreign key mappings among tables and reduce JOIN paths for SQL, we define a more configurable, systematical and structured schema evolution taxonomy. Besides, our provided schema evolution and synthesis framework allows us to explore the schema change on multiple databases easily, while FoodballDB is only limited to one database. Finally, for the seed data selection, compared to Spider, which is commonly used to study text-to-SQL robustness, BIRD features more intricate, realistic, and extensive databases, as well as more complex SQL queries that include keywords often missing in Spider. These distinctions make EvoSchema particularly well-suited for studying how systems adapt to evolving schemas, advancing beyond the simpler or less holistic setups of prior benchmarks.

3.8 Data Statistics

Table 2 provides an overview of the data statistics in EvoSchema, showcasing the various perturbation types applied to the original BIRD dataset. "Column Manipulation" refers to applying the column-level operations on the columns of the original BIRD data; "Table Manipulation" refers to applying the table-level operations on the tables of the original BIRD data. All the perturbed data are obtained by applying column manipulation or table manipulation on the original BIRD dataset. "Manipulated Items" shows the size of the altered columns or the tables. "Manipulated Items/Query" refers to the number of columns or tables modified in the schema

Table 2: Data statistics of EvoSchema. "Original" refers to the original BIRD dataset; "Column Manipulation" refers to applying the column-level operations on the columns of the original BIRD data; "Table Manipulation" refers to applying the table-level operations on the tables of the original BIRD data. "*": the evaluation data for calculating execution accuracy. We synthesize values to reconstruct the database after schema evolution, and filter out those not executable by gold SQL, which results in the smaller size of the evaluation data for calculating execution accuracy.

Data Statistics												
Perturbation Type	Train	Eval	Eval*	Manipulated Items/Table				Manipulated Items/Query			
				Min	Mean	Median	Max	Min	Mean	Median	Max
Original	9426	1534	1068	-	-	-	-	-	-	-	-
				Column Manipulation							
Add Columns	9219	1506	846	1	5.7	3	83	1	5.9	4	43
Remove Columns	9426	1534	1076	1	6.2	2	87	1	6.9	3	70
Remove Col in SQL	9424	1534	-	1	2.5	2	8	1	2.5	2.5	6
Rename Columns	9385	1533	947	1	4.3	3	46	1	4.4	3	46
Split Columns	140	37	37	1	2	2	4	1	2	2	4
Merge Columns	148	44	44	2	3	3	4	2	3	3	4
				Table Manipulation							
Add Tables	9387	1530	1014	-	-	-	-	1	2	2	3
Remove Tables	7212	1171	-	-	-	-	-	1	1	1	1
Rename Tables	9392	1534	1063	-	-	-	-	1	1.5	1	4
Split Tables	9254	1515	824	-	-	-	-	1	2.6	3	5
Merge Tables	6930	1139	569	-	-	-	-	2	2	2	2

for each SQL query, specifically targeting the tables relevant to generating that query. For "Split Tables," "Manipulated Items/Query" represents the number of tables each original table is split into. For "Merge Tables", "Manipulated Items/Query" indicates the number of tables combined into a single table.

4 TRAINING PARADIGM

In our work, we propose a new training paradigm to enhance the model's robustness against different schema evolution. For each *<NLQ, relevant schema, SQL>* triple, we fix the NLQ in the training data, and augment each triple with different schema designs, which may or may not lead to SQL change. Consequently, we obtain multiple triples that can be derived from each of the original triples. We train the model by learning multiple schema designs and SQLs to the original question mappings, which can improve the model's ability to identify the correct relationships among different tables and columns to the question, and can better distinguish the difference among different schema designs. Through this procedure, the model can avoid learning spurious patterns better and therefore enhance the robustness against different schema evolution types.

5 EXPERIMENT SETUP

5.1 Training and Evaluation Settings

Training Setting: We choose four open-source models: Code Llama-7B [25], Mistral-7B [12], Llama 3-8B [7] and SQLCoder-7B [4] and two closed-source models: GPT-3.5 [5] and GPT-4 [22] for our experiments. For these four open-source models, we explore two settings: 1) without perturbation types: the model is trained on the original training data without any perturbation types introduced during training. 2) with perturbation types: the model is trained by merging both the original training data and the perturbation training data. For closed-source models, we only use them for evaluation.

Evaluation Setting: For all the closed-source models and the finetuned open-sourced models, we evaluate them under two settings: 1) without perturbation types: this setting uses the standard, unaltered original evaluation data to evaluate the model performance. 2) with perturbation types: the models are evaluated on data where different perturbations are introduced. By comparing the model performance under these two settings, we can assess how resilient the finetuned models and GPT models are to schema evolution in NL2SQL. This setup provides a comprehensive evaluation of model performance in both standard and perturbed environments, allowing for detailed analysis of robustness and adaptability across different models and schema evolution types.

5.2 Evaluation Metrics

1) Table Match F1: this score is a metric to measure how well the model correctly identifies the relevant tables required to generate a valid SQL query. The F1 score is a harmonic mean of precision and recall, where the precision is the percentage of tables correctly predicted out of all tables predicted by the model and the recall is the percentage of tables correctly predicted out of all the actual tables that should have been selected. The Table Match F1 score combines these two metrics to provide a balanced evaluation, which can assess the ability of text-to-SQL models to correctly identify the required tables from the database schema to form accurate queries. A higher Table Match F1 indicates better performance in selecting the correct tables for the SQL query.

2) Column Match F1: this score is to evaluate how accurately the model identifies the relevant columns required to generate a valid SQL query from a natural language input. Like the Table Match F1, it measures the balance between precision and recall but is applied specifically to the columns of the database. A higher Column Match F1 score indicates better performance in selecting the right columns for the SQL query.

3) Execution Accuracy: this metric measures whether the predicted SQL query can return the correct results as the gold SQL when executing against a database.

5.3 Training and Evaluation Details

We choose Code Llama-7B [25], Mistral-7B [12], Llama 3-8B [7] and SQLCoder-7B [4] as our open-source base models. We fine-tune these models with Huggingface transformers library [31]. For the perturbation training, we merge all the perturbation data and randomly shuffle them as our final training data. We use a learning rate of 2e-5 for training Code Llama, Llama 3 and SQLCoder, and 5e-6 for training Mistral. Our batch size is 4. We train all the models on 4 A100 80GB GPUs and use a cosine scheduler with a 0.03 warm-up period for 6 epochs. We employ FSDP [37] to efficiently train the model. We set the max input length of training as 1024 and the max output length of inference as 500. For inference, we use vllm [31] for batch evaluation, and we set the batch size as 16. We do the inference on an 80G A100 GPU. For closed-source LLMs, we use

[4]https://huggingface.co/defog/sqlcoder-7b-2
[5]https://openai.com/chatgpt/

Table 3: Evaluation on EvoSchema. "w/": the model is trained by merging the original data and all the perturbation training types together; "w/o": the model is only trained on the original training data. The best performance for each model is in bold, and red shows a larger gain. "-": some of the relevant tables are removed so there should be no gold SQL used to calculate the metrics here.

Perturbation Type	Code Llama		Mistral		Llama 3		SQLCoder		GPT-3.5	GPT-4
	w/o	w/	w/o	w/	w/o	w/	w/o	w/		
Table Match F1										
Original	89.77	**90.42**	89.58	**90.62**	**89.96**	89.53	89.69	**90.64**	87.28	88.98
Add Columns	89.73	**90.27**	89.65	**90.03**	89.08	**89.70**	89.30	**90.52**	86.35	88.12
Remove Columns	89.82	**90.24**	89.89	**90.66**	**90.09**	89.82	89.81	**90.54**	87.18	88.87
Rename Columns	**85.28**	85.07	**84.32**	84.27	**83.74**	82.92	**85.32**	84.93	81.73	83.20
Split Columns	83.78	**89.19**	83.78	**88.29**	81.08	**85.14**	86.49	**88.29**	81.44	86.31
Merge Columns	**88.65**	87.23	87.23	**89.72**	**88.65**	86.17	87.23	87.23	83.17	89.36
Add Tables	57.88	**89.50**	57.67	**89.30**	55.11	**88.51**	57.44	**89.38**	83.54	85.79
Remove Tables	-	-	-	-	-	-	-	-	-	-
Rename Tables	88.84	**90.32**	89.40	**90.56**	87.18	**89.14**	89.40	**90.48**	87.02	88.45
Split Tables	71.99	**81.55**	66.12	**80.87**	71.08	**80.12**	72.52	**81.92**	77.52	80.68
Merge Tables	85.29	**87.03**	83.39	**86.91**	81.68	**86.48**	84.80	**86.35**	83.04	86.99
MacroAvg	83.10	**88.08**	82.10	**88.12**	81.77	**86.75**	83.20	**88.03**	83.83	86.68
Column Match F1										
Original	80.66	**81.64**	81.10	**82.36**	**79.13**	78.72	81.52	**81.97**	78.28	80.78
Add Columns	78.26	**80.27**	79.16	**80.18**	75.79	**76.87**	79.09	**80.46**	75.03	78.58
Remove Columns	82.67	**82.75**	83.09	**84.00**	**81.56**	80.69	**83.20**	83.18	80.33	82.55
Rename Columns	76.50	**76.94**	76.35	**76.73**	**72.24**	71.07	76.84	**77.38**	73.40	75.90
Split Columns	71.22	**81.81**	70.24	**80.41**	67.29	**75.04**	74.50	**79.92**	73.59	77.92
Merge Columns	83.19	**83.30**	82.75	**83.41**	82.72	**83.68**	82.64	**83.31**	78.13	88.56
Add Tables	63.81	**81.14**	65.39	**81.09**	59.36	**77.96**	62.91	**81.23**	76.45	79.32
Remove Tables	-	-	-	-	-	-	-	-	-	-
Rename Tables	79.60	**80.91**	80.32	**81.29**	**77.49**	77.46	80.77	**81.79**	77.78	80.04
Split Tables	75.30	**78.45**	73.87	**78.11**	73.81	**73.95**	75.83	**78.59**	74.89	77.41
Merge Tables	65.56	**67.09**	64.12	**67.46**	63.50	**64.40**	65.57	**67.29**	63.23	68.13
MacroAvg	75.68	**79.43**	75.64	**79.50**	73.29	**75.98**	76.29	**79.51**	75.11	78.92

Azure OpenAI API[6]. We use the 2023-12-01-preview version for GPT-4, and 2023-07-01-preview version for GPT-3.5.

5.4 Baselines

We add in-context learning [10] and more advanced method: CHESS [28] as the baselines for comprehensive comparison. In order to test whether the in-context learning can help address the schema evolution issue, we randomly select three examples (each example is an *<NLQ, database schema after evolution, gold SQL after schema evolution>* triple) as the demonstration in the prompt to help the models understand the schema after evolution (Table 4). We also include CHESS, an advanced method for NL2SQL as a baseline. We apply the schema selection (SS) and candidate generation (CG) components developed in their work. For schema selection, we use advanced gpt-4o model to prune the database schema and remove the irrelevant tables and the irrelevant columns in the selected tables, ensuring only the most relevant tables and columns are passed into the model for SQL generation. To ensure a fair comparison with our primary fine-tuning approach, we use a fine-tuned Code Llama model trained without any schema perturbation data as the SQL generation model. This setup allows us to isolate and evaluate the effectiveness of a schema selection and pruning component in addressing schema evolution. The results are shown in Table 4.

[6]https://learn.microsoft.com/en-us/azure/ai-services/openai/reference

6 RESULTS AND ANALYSIS

6.1 Main Results

As Table 3 and Table 5 show, we train Codellama, Mistral, Llama3 and SQLCoder on the original BIRD training data with and without different perturbation types, and evaluate the model on the original BIRD evaluation data and different perturbation types. We observe:

The models trained on different perturbation types are more robust to the schema variation on average, and demonstrate high robustness on the table-level schema evolution. While adding the perturbation data during training leads to a slight Exec Acc (EX) drop for original non-evolved evaluation data, adding, removing and renaming column types, it achieves significantly better results on splitting columns and table-level perturbation types.

By comparing these four models' performance with and without the perturbation data, we observe that for splitting columns, the model trained with perturbation data can achieve up to 5.4 points gain for table match F1, 10.6 points gain column match F1 and 24 points gain for EX; for adding tables, the model trained with perturbation data can achieve up to 33 points gain for table match F1, 18 points gain for column match F1 and 19 points for EX; for splitting tables, the model trained with perturbation data can achieve up to 14 points gain for table match F1, 4.2 points gain for column match F1 and 12 points for EX; for merging tables, the model trained on perturbation data can achieve up to 4.8 points gain on table match F1 and 3 points gain for column match F1. We hypothesize that this is because the perturbation augmented data is particularly beneficial for handling substantial schema changes, but may introduce minor noise in simpler schema changes where the model trained without perturbation data has already maximally learned the patterns. To better understand the slight performance gap under simpler column-level perturbations, we conducted error analysis and case studies to compare models trained with and without perturbed data. We observed two types of errors that lead to this phenomenon: (1) Spurious or missing conditions in the WHERE clause. For instance, given the question "What is the element with the atom ID of TR004_7 in molecule that is not carcinogenic?", the model trained with perturbation ("w/") misses the condition T2.label = '-' in WHERE clause, while the "w/o" model includes it correctly. However, in another case, 'How many transactions were paid in CZK on the morning of 2012/8/26?', the "w/" model introduces an unnecessary WHERE condition: T1.TransactionID BETWEEN 1 AND 1000, which is not part of the gold SQL. (2) Incorrect column selection in SELECT or WHERE clauses. For example, for the question "Among the patients followed at the outpatient clinic, how many of them have a normal level of alkaliphophatase?", the "w/" model predicts T1.Description instead of T1.Admission in WHERE clause, while the "w/o" model selects the correct column. Similarly, in the question "Which group does superhero A-Bomb belong to?", the "w/" model selects T2.team_affiliation instead of the correct T2.race. These examples suggest that while training with perturbed data can improve general robustness, especially beneficial for handling substantial schema changes, it may also introduce minor noise that misleads in condition or column selection under simpler perturbations.

Table 4: Human Evaluation on EvoSchema. "ZS" refers to zero-shot, which prompts models without any examples. "ICL" refers to in-context learning, which prompts models with three demonstration examples. "w/o" means fine-tuning model without perturbation training data; "w/" means fine-tuning model with perturbation training data. Bold color indicates the best performance among each row.

Human Evaluation on EvoSchema					
Perturbation Type	GPT-4		Code Llama		CHESS$_{SS+CG}$
	ZS	ICL	w/o	w/	
Original	62	58	**65**	64	63
Add Columns	59	55	62	61	**66**
Remove Columns	65	61	**66**	63	64
Rename Columns	57	56	57	57	**62**
Split Columns	46	59	41	**62**	49
Merge Columns	68	66	70	**70**	66
Add Tables	56	55	46	**62**	57
Remove Tables	-	-	-	-	-
Rename Tables	58	60	**64**	61	61
Split Tables	57	53	48	**60**	53
Merge Tables	55	57	54	**58**	53
MacroAvg	58	58	57	**62**	59

Table 5: Execution Accuracy on EvoSchema. "w/": the model is trained with all the perturbation types; "w/o": the model is only trained on the original training data.

Exec Acc on EvoSchema										
Perturbation Type	Code Llama		Mistral		Llama 3		SQLCoder		GPT-3.5	GPT-4
	w/o	w/	w/o	w/	w/o	w/	w/o	w/		
Original	58	57	59	58	55	51	58	58	44	47
Add Columns	57	55	56	56	52	49	55	57	43	46
Remove Columns	59	57	60	58	56	53	60	58	45	47
Rename Columns	54	52	55	54	49	47	56	55	43	45
Split Columns	41	**62**	35	**54**	38	**49**	43	**67**	41	46
Merge Columns	70	70	70	70	73	73	66	**82**	61	68
Add Tables	40	**58**	39	**58**	37	**52**	40	**57**	44	48
Remove Tables	-	-	-	-	-	-	-	-	-	-
Rename Tables	56	55	55	56	52	50	56	55	43	47
Split Tables	38	**46**	36	**48**	40	41	43	**49**	40	47
Merge Tables	43	45	45	46	42	44	47	46	37	45
MacroAvg	52	**56**	51	**56**	49	**51**	52	**58**	44	49

Closed-source models are robust to different scheme evolution types in general. As table 3 and 5 show, we compare the model performance on GPT models and four open-source models trained with and without perturbation types. We observe that: the GPT models' performance are relatively stable across different perturbation types compared to the original non-evolved test set. In contrast, fine-tuned open-source models without perturbation training data exhibit significant performance drops—particularly on split columns, add tables, split tables, and merge tables—which introduce larger schema changes. We hypothesize that the stability and robustness of closed-source models stems from broader pretraining exposure and stronger internal schema reasoning capabilities, while the open-source models trained without perturbation types are more sensitive due to limited training on diverse schema variations. This motivates the need to fine-tune open-source models with perturbation training data to improve their generalization under schema evolution. *We notice that comparing the model performance on the open-source LLMs and closed-source LLMs, the models trained with perturbation data have better performance than GPT models on both column-level and table-level perturbation evaluation data.* This indicates that our models trained with perturbation data are more robust than GPT models.

Table-level perturbation has a larger impact than column-level perturbation on the model performance. As Table 3 and 5 show, comparing with the performance on the original evaluation data: adding tables and splitting tables will lead to a significant table match F1 drop; adding tables, splitting tables and merging tables will lead to a significant column match F1 drop. This phenomenon indicates that adding tables or splitting tables easily confuses the models in choosing the correct tables to generate the SQL query. For merging tables, even though the model can correctly choose tables,

it's a bit hard for the model to pick up the correct columns when the columns from different tables go into the same table. While for the column-level performance, there are limited differences with the performance on the original data except for splitting columns. **Reducing table schema complexity is beneficial for model performance.** Compare the model performance on column-level perturbation evaluation and the original evaluation data, adding columns results in a decrease in column match F1, whereas removing columns leads to an increase in column match F1. It indicates simpler table schema is beneficial for models to select columns, as removing columns simplifies the table schema while adding columns makes the table schema more complex.

6.2 Comparison of Different Baselines

As EvoSchema has a large scale of the test set and we need to call GPT-4 and GPT-4o API for in-context learning and CHESS respectively, to save the cost, we randomly select 200 examples for the raw BIRD test set and also from each perturbation type to compare different baselines. We compare GPT-4 zero-shot prompting, GPT-4 3-shot in-context learning, CodeLLama trained with and without perturbation training data and CHESS (with schema selection (SS) and candidate generation (CG)) on our downsampled test set. Since we found that Exec Acc can still make mistakes when different SQL queries produce the same results sometimes even they don't align with the NLQ, or sometimes both the gold SQL and wrong predicted SQL return the empty which may mislead the evaluation, we use human evaluation here for more precise evaluation. As Table 4 shows, compared to GPT-4 zero-shot (ZS), in-context learning (ICL) shows a significant advantage only on the split columns perturbation, while performing slightly better or worse on other types. This suggests that ICL is not consistently effective for handling schema evolution. We hypothesize this is because the demonstration examples in ICL cannot cover the full range of schema and SQL changes; thus, for examples that differ significantly from the demonstrations, ICL offers limited benefit. However, for split columns, where changes commonly involve patterns like name, address, or date splits, the demonstrations generalize better—making ICL more effective in this case. For CHESS, we use GPT-4o—a powerful closed-source model—for schema selection and pruning, and Code Llama without perturbation training (CodeLlama w/o) as the SQL generation model. CHESS achieves the best performance on add columns and rename columns, and significantly outperforms CodeLlama w/o on split columns, add tables, and on average. This highlights the importance of accurate schema selection and pruning in improving SQL generation. However, we also observe that errors at the pruning stage can propagate, leading to degraded performance. Specifically, in merge columns and merge tables cases, CHESS tends to over-prune, omitting relevant schema information and resulting in worse performance than CodeLlama w/o. Finally, we found that fine-tuning CodeLlama with perturbation training data is still needed, since this method gets the best performance among all the baselines on average across all types of evaluation data, and performs significantly better than others on 'split columns', 'add tables', 'split tables' and 'merge tables' types. We applied McNemar's Test [21] to measure the statistical significance of performance differences between our method and each baseline. We computed p-values using the `statsmodels` package, considering differences statistically significant when $p < 0.05$, which indicates that the improvement is unlikely due to random chance. Using this test, we observed our method achieved statistically significant improvements over three key baselines: GPT-4 in-context learning, fine-tuning without perturbed data, and CHESS (all with $p < 0.05$).

Table 6: Perturbation type ablation on EvoSchema. The base model is Code Llama. "both": the model is trained with both column-level perturbation and table-level perturbation types; "w/o table-p": the model is trained without table-level perturbation types; "w/o column-p": the model is trained without column-level perturbation types.

Perturbation Type Ablation						
Perturbation Type	Table Match F1			Column Match F1		
	both	w/o table-p	w/o column-p	both	w/o table-p	w/o column-p
Original	90.73	90.80 (+0.07)	90.04 (-0.69)	81.09	82.15 (+1.06)	80.49 (-0.60)
Add Columns	90.86	90.80 (-0.06)	89.75 (-1.11)	79.63	80.81 (+1.18)	77.29 (-2.34)
Remove Columns	90.72	90.83 (+0.11)	90.48 (-0.24)	83.28	83.85 (+0.57)	82.61 (-0.67)
Rename Columns	85.35	85.38 (+0.03)	84.57 (-0.78)	76.49	77.53 (+1.04)	75.17 (-1.32)
Add Tables	88.95	58.94 (-30.01)	88.57 (-0.38)	79.87	64.11 (-15.76)	79.33 (-0.54)
Remove Tables	-	-	-	-	-	-
Rename Tables	90.54	90.77 (+0.23)	89.29 (-1.25)	81.13	81.51 (+0.38)	79.33 (-1.80)
Split Tables	80.71	73.28 (-7.43)	79.05 (-1.66)	77.41	75.95 (-1.46)	76.30 (-1.11)
Merge Tables	88.72	87.87 (-0.85)	86.83 (-1.89)	68.40	68.26 (-0.14)	67.08 (-1.32)

Table 7: Out of Scope Effect on EvoSchema. The base model is Code Llama. "w/o": the model is trained without perturbation types; "w/": the model is trained on the original data and all the perturbation types; "+ OOS": the model is trained on the original data, perturbation types and two out-of-scope (OOS) perturbation types; "+ OOS FP": The model trained with two OOS perturbation types makes an incorrect prediction on the original data and in-scope perturbation data; "+ OOS TP": The model trained with two OOS perturbation types makes the correct prediction on the two OOS perturbation data; "Tab": the model refuses to predict SQL due to the lack of table information; "Col": the model refuses to predict SQL due to the lack of column information.

Out of Scope Effect										
Perturbation Type	Table Match F1			Column Match F1			+ OOS FP		+ OOS TP	
	w/o	w/	+ OOS	w/o	w/	+ OOS	Tab	Col	Tab	Col
Original	89.77	90.42	82.98 (-7.44)	80.66	81.64	75.43 (-6.21)	7.11	0.65	-	-
Add Columns	89.73	90.27	86.07 (-4.20)	78.26	80.27	77.00 (-3.27)	4.25	0.40	-	-
Remove Columns	89.82	90.24	82.24 (-8.00)	82.67	82.75	75.90 (-6.85)	7.56	0.72	-	-
Remove Col in SQL	-	-	-	-	-	-	5.02	-	-	84.03
Rename Columns	85.28	85.07	80.20 (-4.87)	76.50	76.94	73.04 (-3.90)	4.44	0.20	-	-
Add Tables	57.88	89.50	88.78 (-0.72)	63.81	81.14	80.71 (-0.37)	0.33	0.07	-	-
Remove Tables	-	-	-	-	-	- -	-	1.62	83.86	-
Rename Tables	88.84	90.32	86.36 (-3.96)	79.60	80.91	78.06 (-2.85)	3.52	0.39	-	-
Split Tables	71.99	81.55	81.07 (-0.48)	75.30	78.45	78.02 (-0.43)	0.26	0.07	-	-
Merge Tables	85.29	87.03	82.18 (-5.15)	65.56	67.09	63.59 (-3.50)	4.65	0.35	-	-

6.3 Influence of Perturbation Types

We explore the effect of the column-level perturbation types and table-level perturbation types. As Table 6 shows, we train the model with both column-level and table-level perturbation types, and compare it with the model trained without column-level perturbation types and without table-level perturbation types. From our

experiments, we found that without training on table-level perturbations, the model performance can be slightly better than the model trained with both column-level and table-level perturbation types on column-level perturbation types, while can lead to a significant performance drop on the table-level perturbation types. This indicates that the table-level perturbation data has a limited effect on the column-level perturbation types while having a huge impact on the table-level perturbation types. When looking at the model trained only on table-level perturbation types, we found that the model performance on both column-level and table-level perturbation types dropped. This indicates that the column-level perturbation types can still benefit the training.

Table 8: Irrelevant tables effect. "w/": the model is trained with all the perturbation types; "w/o": the model is only trained on the original training data; "w/o+": the model is only trained on the original training data, but for the input table schema, we also add irrelevant tables.

Add Irrelevant Tables Effect						
Perturbation Type	Table Match F1			Column Match F1		
	w/o	w/o+	w/	w/o	w/o+	w/
Original	89.77	87.65	**90.42**	80.66	79.24	**81.64**
Add Columns	89.73	86.35	**90.27**	78.26	75.31	**80.27**
Remove Columns	89.82	87.30	**90.24**	82.67	80.74	**82.75**
Rename Columns	**85.28**	81.90	85.07	76.50	73.28	**76.94**
Add Tables	57.88	88.01	**89.50**	63.81	79.51	**81.14**
Remove Tables	-	-	-	-	-	-
Rename Tables	88.84	86.84	**90.32**	79.60	78.47	**80.91**
Split Tables	71.99	67.27	**81.55**	75.30	70.39	**78.45**
Merge Tables	85.29	83.56	**87.03**	65.56	63.59	**67.09**

6.4 Influence of Out-of-scope Types

We evaluate both in-scope and out-of-scope scenarios. In in-scope settings, schema changes may or may not alter the gold SQL. Out-of-scope cases involve two special perturbations: (1) *Removing columns used in the gold SQL*, and (2) *Removing tables used in the gold SQL*. In both cases, the schema lacks critical information, and the model is expected to abstain from generating a query.

To assess their impact, we train a model on a combined dataset that includes both out-of-scope and in-scope perturbation types, along with the original training data. We compare this model to others trained only on the original or in-scope data. As shown in Table 7, incorporating out-of-scope types results in performance degradation across both original and in-scope evaluation sets.

Error analysis reveals that the model trained with out-of-scope data tends to make more conservative predictions, sometimes abstaining even when the gold SQL is valid. Further analysis shows that the false positive (FP) rate closely matches the performance drop between models with and without out-of-scope training, confirming that increased conservatism is the main cause. Additionally, for the out-of-scope perturbations, the TP is only around 84%, which indicates that the model still has a 16% chance to make a prediction even when there should not be an SQL.

6.5 Influence of Irrelevant Tables

We observed that the model trained with perturbation types demonstrates significant robustness to table-level perturbations, such as adding and splitting tables. Upon analyzing the errors, we found that models trained without perturbation types tend to predict SQL queries that join all available tables, even when some tables are irrelevant to the NLQs and SQLs. We hypothesize that this occurs because during training without perturbations, the model only sees relevant table schemas, causing it to learn spurious patterns that always try to join all the input tables.

To explore whether simply adding irrelevant tables could yield similar performance to models trained with perturbation data, we conducted an experiment where we trained CodeLlama on BIRD. As shown in Table 8, adding irrelevant tables led to similar performance on "Add Tables" perturbation type. but it caused a performance drop on other perturbation types. This suggests that combining all perturbation data is necessary to train a more robust model.

Table 9: Intra-database Effect. This experiment emphasizes that the training and evaluation occur within the same database, instead of across databases.

Intra-database Effect				
Perturbation Type	Table Match F1		Column Match F1	
	w/o	w/	w/o	w/
Original	87.24	**87.43**	79.54	**80.89**
Add Columns	87.14	**87.43**	76.36	**78.92**
Remove Columns	**87.29**	87.27	81.14	**81.29**
Rename Columns	85.71	**86.43**	77.45	**79.09**
Add Tables	61.13	**83.95**	66.11	**78.57**
Remove Tables	-	-	-	-
Rename Tables	86.33	**86.67**	79.44	**79.96**
Split Tables	71.82	**78.52**	75.09	**77.42**
Merge Tables	85.11	**87.44**	71.43	**74.72**

6.6 Influence of Intra-DB and Cross-DB

We hypothesize that a model trained on the same databases may not only learn schema evolution patterns but also become familiar with specific table and column names. To test this, we split the BIRD training data into train/test sets to ensure that each database in the test set also appears in the training set. We use Code Llama as the base model. The results in Table 9 show that, for most perturbation types, the model's performance improves more compared to the cross-database scenario in Section 6.1, which verifies our hypothesis.

7 CONCLUSION

In conclusion, we formulate the critical challenge of schema evolution in adaptive text-to-SQL systems and introduce EvoSchema, a comprehensive, diverse and unique benchmark designed specifically to study and address this problem. We developed a structured taxonomy of schema evolution types, enabling the synthesis of realistic schema designs through column-level and table-level perturbations. Using this taxonomy, we construct an evaluation benchmark to rigorously assess model robustness under schema changes and also introduce a novel training paradigm that augments existing *<NLQ, relevant schema, SQL>* triples with diverse schema designs for training to improve robustness against schema evolution.

ACKNOWLEDGMENTS

The authors would like to thank colleagues from the OSU NLP group for their insightful discussions and constructive suggestions and all anonymous reviewers for their thoughtful comments.

REFERENCES

[1] Andre Becklas. 2018. FIFA World Cup: All the results from World Cups. Kaggle (2018). https://www.kaggle.com/datasets/abecklas/fifa-world-cup

[2] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness. In The Eleventh International Conference on Learning Representations. https://openreview.net/forum?id=Wc5bmZZU9cy

[3] Anthony Cleve, Maxime Gobert, Loup Meurice, Jerome Maes, and Jens Weber. 2015. Understanding database schema evolution: A case study. Science of Computer Programming 97 (2015), 113–121.

[4] Daiga Deksne and Raivis Skadiņš. 2022. Virtual Assistant for Querying Databases in Natural Language. In Proceedings of the Future Technologies Conference. Springer, 555–564.

[5] Julien Delplanque, Anne Etien, Nicolas Anquetil, and Olivier Auverlot. 2018. Relational Database Schema Evolution: An Industrial Case Study. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). 635–644. https://doi.org/10.1109/ICSME.2018.00073

[6] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. 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. https://doi.org/10.18653/v1/2021.naacl-main.105

[7] Abhimanyu Dubey and et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[8] Jonathan Fürst, Catherine Kosten, Farhad Nooralahzadeh, Yi Zhang, and Kurt Stockinger. 2025. Evaluating the Data Model Robustness of Text-to-SQL Systems Based on Real User Queries. In EDBT. 158–170. https://doi.org/10.48786/edbt.2025.13

[9] Chang Gao, Bowen Li, Wenxuan Zhang, Wai Lam, Binhua Li, Fei Huang, Luo Si, and Yongbin Li. 2022. Towards Generalizable and Robust Text-to-SQL Parsing. In Findings of the Association for Computational Linguistics: EMNLP 2022, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 2113–2125. https://doi.org/10.18653/v1/2022.findings-emnlp.155

[10] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. Proceedings of the VLDB Endowment 17, 5 (2024), 1132–1145.

[11] Andrea Hillenbrand and Uta Störl. 2021. Managing Schema Migration in NoSQL Databases: Advisor Heuristics vs. Self-adaptive Schema Migration Strategies. In International Conference on Model-Driven Engineering and Software Development. Springer, 230–253.

[12] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[13] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. 2021. Wilds: A benchmark of in-the-wild distribution shifts. In International conference on machine learning. PMLR, 5637–5664.

[14] Kunal Kumar and S. K. Azad. 2017. Database normalization design pattern. In 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON). 318–322. https://doi.org/10.1109/UPCON.2017.8251067

[15] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? arXiv preprint arXiv:2406.01265 (2024).

[16] Guoliang Li, Xuanhe Zhou, and Xinyang Zhao. 2024. LLM for Data Management. Proceedings of the VLDB Endowment 17, 12 (2024), 4213–4216.

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

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

[19] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? arXiv preprint arXiv:2408.05109 (2024).

[20] Pingchuan Ma and Shuai Wang. 2021. MT-teql: evaluating and augmenting neural NLIDB on real-world linguistic and schema variations. Proc. VLDB Endow. 15, 3 (nov 2021), 569–582. https://doi.org/10.14778/3494124.3494139

[21] Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. Psychometrika 12, 2 (1947), 153–157.

[22] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[23] Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. 2022. Towards Robustness of Text-to-SQL Models Against Natural and Realistic Adversarial Table Perturbation. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 2007–2022. https://doi.org/10.18653/v1/2022.acl-long.142

[24] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. 2009. Dataset Shift in Machine Learning. The MIT Press.

[25] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL] https://arxiv.org/abs/2308.12950

[26] Yewei Song, Saad Ezzini, Xunzhu Tang, Cedric Lothritz, Jacques Klein, Tegawendé Bissyandé, Andrey Boytsov, Ulrick Ble, and Anne Goujon. 2024. Enhancing Text-to-SQL Translation for Financial System Design. In Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice. 252–262.

[27] Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring Chain of Thought Style Prompting for Text-to-SQL. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 5376–5393. https://doi.org/10.18653/v1/2023.emnlp-main.327

[28] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755 [cs.LG] https://arxiv.org/abs/2405.16755

[29] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[30] Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust Text-to-SQL Generation with Execution-Guided Decoding. arXiv:1807.03100 [cs.CL] https://arxiv.org/abs/1807.03100

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs.CL] https://arxiv.org/abs/1910.03771

[32] 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, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425

[33] Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the Text-to-SQL Capability of Large Language Models: A Comprehensive Evaluation. arXiv:2403.02951 [cs.CL] https://arxiv.org/abs/2403.02951

[34] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis. arXiv preprint arXiv:2401.10506 (2024).

[35] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. In The 2023 Conference on Empirical Methods in Natural Language Processing. https://openreview.net/forum?id=oeZiXoCHgq

[36] Tianshu Zhang, Changchang Liu, Wei-Han Lee, Yu Su, and Huan Sun. 2023. Federated Learning for Semantic Parsing: Task Formulation, Evaluation Setup, New Algorithms. arXiv:2305.17221 [cs.CL] https://arxiv.org/abs/2305.17221

[37] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can

Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC] https://arxiv.org/abs/2304.11277

[38] Alex Zhuang, Ge Zhang, Tianyu Zheng, Xinrun Du, Junjie Wang, Weiming Ren, Stephen W Huang, Jie Fu, Xiang Yue, and Wenhu Chen. 2024. StructLM: Towards Building Generalist Models for Structured Knowledge Grounding. arXiv preprint arXiv:2402.16671 (2024).