



The University of Western Ontario
Department of Computer Science

Learning Text-to-SQL with Large Language Models: A Survey-Based Project Report

Group Members:

Annabel Irani

Otilia Pasculescu

Darwish Chahbar

Maximus Taillon

Martin Tzanev

Databases II (Advanced Databases)

Project Report

December 7, 2025

Contents

1	Introduction	2
2	Background and Project Goals	2
3	Paper 1: Survey on LLM-Based Text-to-SQL	4
3.1	Literature Review	4
4	Paper 2: Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation	6
4.1	Literature Review and Idea Extension	6
4.2	Literature Review	6
5	Paper 3: SQLPrompt: In-Context Text-to-SQL with Minimal Labeled Data	8
5.1	Literature Review	8
6	Paper 4: DR.SPIDER: A Diagnostic Evaluation Benchmark Towards Text-to-SQL Robustness	10
6.1	Literature Review	10
6.2	Reproduction Findings	12
6.3	Idea Extension	14
7	Paper 5: Natural SQL: Making SQL Easier to Infer from Natural Language Specifications	15
7.1	Literature Review	15
7.2	Reproduction Plan	16
7.3	Reproduction Findings	17
7.4	Idea Extension	19
8	Conclusion	19

Abstract

This project investigates how large language models (LLMs) translate natural language questions into SQL queries in modern Text-to-SQL systems. Working within the CS4411 Data Management Systems context, our group studies five research papers covering prompt engineering, benchmark evaluations, and system design for LLM-based Text-to-SQL. We focus especially on two papers for which we design small-scale reproduction plans, examining the impact of structured prompts, schema linking, and reasoning strategies on accuracy over benchmark datasets such as Spider. Finally, we propose idea extensions that address open challenges identified in the literature, including context limitations, complex schemas, and the integration of domain knowledge.

1 Introduction

Text-to-SQL aims to allow users to query relational databases using everyday natural language, automatically converting SQL queries, which allows users to access relational databases without any prerequisite knowledge of query languages. In the past, we relied heavily on encoder-decoder models trained on large text-SQL corpora, but this method required extensive labeled data, struggled to generalize unseen schema and often failed on complex queries involving new or evolved linguistics. Recently, large language model has begun making new opportunities for Text-to-SQL by enabling few-shot prompting, schema-aware reasoning and more flexible query generation.

This project investigates how LLMs can be used to reliably convert natural language into SQL queries by combining theoretical understanding with research and experimentation. The goal of this research paper is (1) to review recent research on LLM based Text-to-SQL methods, (2) reproduce the experimental results from selected papers, and (3) to use critical, analytical and creative thinking to extend these ideas through the design and testing of additional strategies we present. This will deepen our understanding of the topic overall, the current research done and what else is left for Text-to-SQL research.

This work contributes to the effort of making databases more accessible to non-technical users. Through systematic analysis, reproduction and methodological extension, the project demonstrates how LLMs can support natural language interfaces and identifies practical strategies for improving their reliability in real-world settings.

The paper is organized as follows: Section 2 summarizes background and project goals. Sections 3–5 briefly review the literature and provide short analyses. Sections 6 and 7 present detailed analyses, reproduction plans, and idea extensions for our first two focal papers. Section 8 summarizes our findings and conclusions.

2 Background and Project Goals

All development of Text-to-SQL with LLM up to now is how we have OpenAI in the present day. Starting in 1967, Eliza was the first known form of natural language processing in the form of the world’s first chat bot (“The history, timeline, and future of

LLMs” 2023). Eliza would be known to be the first major step towards understanding natural language. In 1997, Long Short Term Memory networks were introduced showing the ability to process sequential data(“The history, timeline, and future of LLMs” 2023). After further development coming from Stanford and Google, Text-to-SQL with LLMs now takes the form of AI that is accessible to the general public such as Claude, Chat GPT and AutoPilot (“The history, timeline, and future of LLMs” 2023). After reading over the provided readings, they mention that most previous work on text-to-SQL focuses on extracting the question-to-SQL patterns and generalizing them by training an encoder-decoder model with Text-to-SQL corpus. In recent years, LLMs have become the new support of Text-to-SQL. The readings also mention some current problems still arising such as how to prompt LLM to generate correct SQL queries, namely prompt engineering. All research and testing with Text-to-SQL LLMs is done with benchmark datasets, the one mentioned in the papers provided is Spider with grades roughly around 85% (“Text-To-SQL Empowered by Large Language Models: A Benchmark Evaluation.” Dawei Gao. <https://www.vldb.org/pvldb/vol17/p1132-gao.pdf>.) As LLMs continue to advance, Text-to-SQL systems are becoming more accurate and capable of allowing users to query databases through language.

Text-to-SQL allows people to interact with databases using everyday language by turning questions into SQL queries. This makes databases easier to use for people who don’t know SQL and also enables building tools that can answer questions or analyze data automatically. Earlier approaches, such as PICARD, UnifiedSKG, and RESDSQL-3B + NatSQL, trained models on a lot of paired text-and-SQL examples to get accurate results. Although these methods work well, they require a lot of data and computing power, and do not always adapt easily to new databases.

Large language models (LLMs) like GPT-3, PaLM, and ChatGPT have made this easier through few-shot prompting, where the model learns from just a few examples in the prompt. Methods like SQLPrompt improve this further by using techniques like execution-based consistency checking and mixing different prompts or models to get better SQL results. These approaches reduce the need for huge datasets, handle new questions better, and make Text-to-SQL systems more reliable. Our project will study these methods, reproduce their results, and explore ways to improve LLMs for turning natural language into SQL.

This project builds and uses recent research in Text-to-SQL and LLMs. Papers like SQLPrompt show how few-shot prompting and techniques like execution based consistency can improve SQL generation from natural language. Other studies, including PICARD, UnifiedSKG, and RESDSQL, focus on fine tuning models with large datasets. By reviewing these papers and reproducing their results, we can link theory to practice and explore how to make LLMs more accurate and reliable for converting everyday language to SQL queries.

The primary goal of this project is to explore how large language models (LLMs) can translate natural language queries into structured SQL statements, thereby making databases more accessible to non-technical users. Throughout this project, our aim is to develop both a theoretical understanding as well as practical experience with LLM-based Text-to-SQL systems by not only exploring and understanding current research and experimentation but also by recreating it and possibly contributing with our own findings and discoveries. Our team seeks to understand how LLMs interpret and con-

vert everyday language into database queries, investigate techniques that enhance model performance, and evaluate prompt-based and few-shot learning techniques which improve query accuracy and usability.

To achieve our goals, we will follow this course of action. First, we will read research papers that clarify the current findings and discoveries on the topic of Text-to-SQL with LLMs. Second, we aim to reproduce the research. Third, we plan to expand the research to find out how LLMs can be applied to make databases easier to access through natural language.

In detail, our first step is to review recent research papers and benchmark datasets, such as Spider, to understand and identify the major challenges and the current evaluation methods in Text-to-SQL. We have provided three papers; then we will expand our research by finding more papers related to this topic to find more current research. This will help us achieve our goals and objectives, since after reading the articles ourselves, we will have a better understanding of where the research on this topic stands and how we will be able to evaluate it.

Our second step is to reproduce the selected experimental results to gain insight into the methodologies and outcomes discussed in previous studies. Our aim here is to expand and deepen our knowledge and understanding, making sure that we fully understand the topic and how the research contributed to the advancement of Text-to-SQL with LLMs. This will help us achieve our goals and objectives by allowing us to gain detailed knowledge within the topic and truly master it. It also allows us to learn for ourselves how the research was conducted, putting us in a great position to expand on it.

Our third step is to design and test prompts for LLM-based Text-to-SQL generation, analyze and compare model output using established performance metrics, and summarize best practices from the literature. This could involve applying concepts to new datasets or tweaking existing algorithms. This will help us achieve our goals and objectives by integrating and applying our knowledge on the topic. By extending the research ideas that have already been done, we will deepen our understanding of how LLMs can be applied to make databases easier to access through natural language, which is the ultimate goal for our group.

3 Paper 1: Survey on LLM-Based Text-to-SQL

3.1 Literature Review

“A Survey on Employing Large Language Models for Text-To-SQL Tasks” provides a comprehensive overview of how recent LLMs are being put to use to convert natural language into SQL queries. This topic is important as relational databases require in-depth SQL knowledge, which creates a barrier for many users who don’t have experience in that sector. This paper reviews LLM-based Text-To-SQL methods, compares prompt engineering vs. fine tuning, summarizes benchmarks and metrics and finally proposes a conclusion. The authors’ main contribution is a systematic taxonomy of existing methods and providing an in-depth analysis of LLM-based Text-to-SQL pipelines and finally reinforcing practical utility through concise takeaways and conclusions.

The increasing adoption of LLM-based Text to SQL approaches is driven by several factors. There has been evidence for enhanced performance, as LLM based methods have improved the state-of-the-art performance and demonstrate remarkable capabilities. LLMs, more specifically prompt-engineering, have shown great adaptability and generalization ability making these methods easily transferable between settings with little to no training. Additionally, they hold a promising future for advancements including higher quality results, more optimization and fine-tuning.

The authors categorize current methods into two main streams. Prompt-engineering and fine-tuning. For prompt engineering, authors have broken this into pre-processing, inference, and post-processing. Pre-processing involves constructing prompts, including layouts, schema descriptions and sample data. Inference then happens to generate the corresponding SQL and includes the reasoning workflow, such as Chain-Of-Thought, least-to-most, decomposition and few-shot demonstrations. Finally, post processing refines the output for self-correction, consistency checks and SQL validation to improve stability and overall performance.

After outlining the stages involved in prompt engineering, the authors turn to fine-tuning as the second central method for improving text-to-SQL performance. Fine tuning involves training a model directly on Text-to-SQL datasets like Spider, or BIRD to improve its SQL generation accuracy, schema linking, query planning, reasoning, and stability. Fully fine-tuning and parameter efficient fine-tuning are two streams of training methods, but PEFF is preferred as it is less prone to catastrophic forgetting. Fine-tuning is useful and crucial as open source LLMs like LLaMA, Code Llama often underperformed compared to closed models like GPT-4. SQL is only a tiny portion of their pretraining data. Fine-tuning methods include supervised fine-tuning where a model is trained on natural language, then turned into SQL pairs, Low rank adaptation (LoRA) which has smaller training footprints but strong gains and finally, multi-objective training which includes SQL generation, schema linking, SQL correction, and SQL planning. Fine-tuning is valuable when privacy is a concern. It enables the organizations to adapt an open-source model without sending their database schema to external APIs.

The survey also reviews the benchmarks and evaluation metrics used to assess LLM-based Text-to-SQL systems. Classic datasets, prior to the rise of LLMs include WikiSQL and Spider 1.0. These made crucial advancements and are still the top choice for many. In the era of LLMs, benchmarks include ScienceBenchmark, BIRD, Dr. Spider and more. These benchmarks bridge may gaps and solve problems of previous datasets. They provide a more realistic test of model robustness. To further evaluate the system performance in the real world, the survey highlights metrics such as Exact set match accuracy (EM), Execution Accuracy (EX), Test-suite Accuracy (TS), Valid Efficiency score (VES) and ESM+. These benchmarks and metrics show both the strong potential of LLM-based approaches and the remaining challenges involving schema understanding, robustness and real-world deployment.

The survey utilized 12 closed source LLMs and 16 open source LLMs as base models. The main advantages for closed source models are tied to their large-scale pre-trained corpus and the huge parameter size. Because of their large size, closed source models are difficult to deploy independently and most of their methods use API calls to access these models. At the time this article was written, there was no closed-source

LLM specifically designed for SQL code generation. This could be because SQL requires privacy information in different business data processing which is difficult to conform to a pre-trained LLM, additionally, closed-source models may not be as efficient as the general code generation model. Open-source models on the other hand are often deployed in the private domain with specific hardware support given their, easier to handle, parameter scale. These are seen as baselines for experiments when used by most text-to-SQL methods based on prompt engineering. Current Text-to-SQL work prefers the GPT series closed source models and as for open source, the preferred are DeepSeek, Code Llama, and Qwen. Until now, open-source models have been the most used and preferred.

The authors held an investigation to analyze the practical value and the applicability of each method. They evaluated performance based on three benchmarks, Spider 1.0, the most established benchmark, BIRD, which introduces real-world obstacles, and Spider 2.0, extending the scope to enterprise level workflows, more complex multi-database environments, and a broader range of operations. The results conclude that for Spider 1.0, all of the methods in the analysis passed with most achieving rates of 80% or greater. Modern LLMs have effectively solved the basic Text-to-SQL tasks, confirming our confidence in their ability to handle challenges. For BIRD, it has proven that both large closed-source LLMs and fine-tuned open-source models have a significantly smaller gap than what it once was achieving competitive, SOTA performances. The traits that got them there are inference-time scaling, generating multiple SQL statements, self-correcting and using consistency and reliability. Second, they use enhanced schema-linking by either expanding the search space or by including more schema columns which in turn increases likability of producing a correct answer. Spider2.0, demonstrated the limitations that open-source models still have despite their significant progress. Agent-based methods consistently outperform non-agent approaches since real world text-to-SQL requires diverse handling, understanding complex syntax, as well as the need to integrate external knowledge.

Although there has been significant improvements and advancements in text-to-SQL tasks, there are still many substantial challenges across data quality, privacy, schema complexity and domain adaption. Error analysis shows that schema linking and JOIN mistakes are the biggest source of errors.

4 Paper 2: Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation

4.1 Literature Review and Idea Extension

4.2 Literature Review

This paper addresses the previous lack of a benchmark to assess Text-To-SQL systems. It introduces a system that analyzes many different factors and tests. It talks about the five prompt representations which are basic, text representation, openAI demonstration, code representation, and Alpaca SFT. The paper also offers a new benchmark that can be used to train Text-To-SQL LLM models called DAIL-SQL. Overall it does its best

to outline ways in which Text-To-SQL LLM's can be optimised to be effective and efficient.

The main methods and techniques used by the paper are the previously mentioned prompt representations, and prompt components, example selection methods, example organization styles, DAIL-SQL. As previously mentioned, there are five prompt formats each of which is similar but varies in three categories: instructions, rule implications, and foreign keys. With some representations having more or less of each. The goal was to use the benchmark to test different Text-To-SQL models and the results suggest that all representations are good but some are better for certain models. For example, GPT-4 performs better under basic representation however GPT-3.5 performs better under OpenAI demonstration prompts. Along with prompt representation, the paper also examines prompt components which are instructions, rule implication and foreign key. Where instructions are just natural language and rule implication is natural language with guidance or rules implied. The benchmark finds that rule implication is one of the most effective ways to improve performance in a Text-To-SQL model as adding rules in the prompt can improve performance by up to 6% over a basic prompt with only instruction and no rule. Additionally, adding a foreign key in the prompt has potential to increase performance by up to 3% but is unstable depending on the model.

As for example selection methods, there are four kinds. Those being random, question similarity selection, masked question similarity selection, and query similarity selection. The paper proposes that none of these four are optimal alone so some combination should be used which is where they propose their new example selection technique which is a critical part of DAIL-SQL called DAIL selection. As opposed to other techniques this one doesn't just examine SQL similarity or question similarity but it does both at the same time. This leads to much higher accuracy scores. The second example method is the choice of example organization style. The two organization styles commonly used are Full-Information, which maintains everything including schema, question and SQL queries when passing an prompt or example to an LLM. The second type of organization is SQL only organization which only passes SQL query examples and prompts to the LLM. The benefit to full example organization is that it gives much more information resulting in higher accuracy but demands high performance whereas with SQL-Only the performance is much greater with less accuracy. The paper once again provides a new solution for example organization which is called DAIL organization. DAIL organization is a mix between the two approaches, it sends only the question and the SQL queries. This results in performance nearly as good as full information and outperforms SQL only organization in terms of accuracy while having much better performance than full information.

These two DAIL methods of example selection and organization are used in the creation of the DAIL-SQL benchmark. The DAIL-SQL benchmark is not a model in itself but is used to optimize a model for accuracy and token usage. They utilize a code prompt representation with the DAIL selection and DAIL organization methods to train the GPT-4 model to a record 86.6% accuracy.

Overall, the paper does a good job of creating and highlighting a comprehensive benchmark that is backed by strong results such as the Spider leaderboard score. It specifically talks about the success of using DAIL-SQL and GPT-4 however, the paper also highlights its effect on other prominent Text-To-SQL LLM's. Additionally, it

highlights two new approaches to example organization and example selection. One potential issue which is highlighted in the paper is that for certain weaker LLM's the results are not as consistent, and overall the performance varies across most models. However, for certain models such as GPT-4 it provides very high performance and accuracy.

5 Paper 3: SQLPrompt: In-Context Text-to-SQL with Minimal Labeled Data

5.1 Literature Review

Introduction

The authors introduce the idea of “SQLPrompt”, a few shot prompting approach for Text-to-SQL. The approach is comprised of execution-based consistency decoding and execution error filtering. They employ different prompt designs (“MixPrompt”) and LLMs (“MixLLMs”) to obtain a diverse set of LLM outputs. “MixPrompt” applies different prompt designs which leads to more diverse SQLs. “MixLLMs” assumes that different LLMs learn different sets of knowledge and hence yield different outcomes. They then intend to extrapolate the answer from the consistent answer across different prompt designs and LLMs. The main idea is that LLMs can produce multiple answers queries from one question, so to derive the final answer, we choose by selecting from the set of different answers.

Methods

Problem setup for Text-to-SQL

Let q denote natural language query and D_q denote the database information. The task is to convert q into SQL. The database $D_q = S, K_p, K_f$, holds information about the database schema, S , the primary keys, K_p , and the foreign keys, K_f . S could have multiple tables T_t : $S = T_1, T_2, \dots, T_t, \dots$, and each table T_t has table name N_t , column names c_j and column data types t_j : $T_t = N_t, (ct_1, tt_1), (ct_2, tt_2), (ct_j, tt_j), \dots$. Primary keys K_p uniquely identifying rows of each table, and foreign keys K_f join multiple tables.

Prompt design: database schema and content and primary/foreign keys

The prompt is comprised of database schema, primary and foreign keys, and the database content. The idea is that the design of the prompt impacts the number of different answers generated. This is where SQLPrompt then introduces the idea of MixPrompt, to make sure the LLMs generate multiple diverse answers. So, they present the information in different formats, with the intentional goal of making them different from each other to encourage diverse outputs.

Concise prompts specify information in a table; it describes the structure of the table in a clear way. On the other hand, it can be harder for LLMs to understand the syntax. Verbose prompts describe databases in a way that is understandable to humans and put an emphasis on whatever LLMs would need to know. When giving LLMs these different formats, it makes it so that the outputs are diverse, and therefore have a

better idea of what query is correct. Refinement based on execution-based consistency with MixPrompt and MixLLMs. The goal is to derive diverse outputs, so the authors use two procedures, MixPrompt and MixLLM. MixPrompt: applies various prompts designs to encourage diverse SQL outputs. Take $F = f_1, f_2, \dots$ as a collection of prompt design functions. Say f_1 is verbose, f_2 is concise. When they fix the LLMs, they have MixPrompt with the following prediction objectives:

$$p(\text{sql}|\text{LLM}, q) = \sum_f p(\text{sql}|\text{LLM}, f, q)p(f)$$

Figure 1

Then evenly mix the prompts, so each prompt format contributes an equal amount. So, $p(\text{sql}|\text{LLM}, f, q)$ is the sampling probability of generating sql. Refinement procedures An algorithm is used after for refinement based on execution and consistency. The algorithm derives many SQL answers using different prompts, it implements them, then filters out all of the answers so that the final SQL answer is the one that is the most consistent across all results. MixLLMs: instead of using one LLM, to increase diversity, use a mixture of LLMs, as to not rely on one LLM for all answers. By combining outputs across multiple LLMs, a higher chance of diversity is achieved. So taking the algorithm used, they include then for each LLM used:

$$p(\text{sql}|q) = \sum_{\text{LLM}} \sum_f p(\text{sql}|\text{LLM}, f, q)p(f)p(\text{LLM})$$

Figure 2

The answer is then derived from different prompt designs and different LLMs.

Experiments

Tasks and datasets: The authors consider Spider, containing 7000 training samples across 166 databases and 1034 evaluation samples across 20 databases. **Models:** They see the performance of SQLPrompt using different variants of PaLM. PaLM FLAN 540B, PaLM62B, and PaLM FLAN 62B. They applied quantization to the models. This reduces the precision but increases the efficiency. **Fine-tuning baselines:** PICARD, RASAT, and RESDSQL are all the models they used as baselines to compare with SQL-Prompt. **Evaluation:** The authors consider two evaluation metrics: execution accuracy (EX) and test-suite accuracy (TS). Where EX measures if SQL execution outcome is the same as the actual ground truth and TS tests each query by making sure the output is the same across all. However, they do not use exact match evaluation, since two SQLs can be true for one query.

Results

The results show that SQLPrompt is strong in text-to-SQL performance. Mix-Prompt adds diversity to the prompt designs, which in turn improves execution accuracy. MixLLMs adds the diversity of having multiple LLMs which also improves accuracy. Results show that SQLPrompt outperforms ChatGPT by an increase of 7%

for execution accuracy and 8.1% for test suite accuracy. There is a 2% improvement over single prompt with MixPrompt. There is a 3% improvement over single LLM in test suite accuracy with MixLLMs. The authors specified that all components in SQLPrompt (prompt design, execution-based consistency decoding, MixPrompt, and MixLLMs) seem to be useful, having around 2-15% improvements. They even noted that without consistency decoding, the performance decreases by 13.6%; so, the consistency decoding contributes by 13.6%. Without execution error filtering, there is a decrease in results by 14.5%.

6 Paper 4: DR.SPIDER: A Diagnostic Evaluation Benchmark Towards Text-to-SQL Robustness

6.1 Literature Review

The research paper I decided to review was one I found online and it came from students studying at Ohio State University. It is titled: DR.SPIDER: A DIAGNOSTIC EVALUATION BENCHMARK TOWARDS TEXT-TO-SQL ROBUSTNESS and was written in 2023. The reasoning for this paper is confirmed in the abstract, and that reason is these students recently studied that recent experiments revealed that text-to-SQL models are vulnerable to task-specific perturbations. Perturbations are modifications to the input data to test a model's robustness and how well it handles variations. They claim a controversy with previous text-to-SQL testing and results is the fact that the results are obtained in the setting where test data are created with the same distribution as training data. The students claim that this setting prevents the evaluation of model robustness, especially when the data contain spurious patterns that do not exist in the wild. How these students wanted to test the robustness of text-to-SQL using a comprehensive Diagnostic Robustness evaluation benchmark based on Spider, which is a cross-domain text-to-SQL benchmark, to diagnose the model robustness, that they called Dr.Spider. They used a variety of testing pools for their 17 perturbations which included perturbations from databases, natural language questions and SQL queries to get a wider scope of results. They also claimed to have used pretrained language models to simulate human behaviours in creating natural questions. They evaluated the text-to-SQL models on their benchmark and they found that the experiments demonstrate that although the models achieve good performance on the original data, they struggle to have consistently correct predictions in our robustness sets. Even the most robust model suffers from a 14.0% performance drop overall and a 50.7% performance drop against the most challenging perturbation.

Now going into the data creation of their experiment, they say that Dr.Spider aims to comprehensively evaluate the robustness of models with perturbations on each component of the text-to-SQL task. Of the 17 different tests they used, Dr.Spider contained 3 database perturbation test sets, 9 natural language question perturbation test sets, and 5 SQL perturbation test sets to simulate various task-specific phenomena. Each test set contained parallel pre-perturbation and post-perturbation data to measure model robustness against the perturbation. In total, Dr.Spider contained 15K pairs of pre-perturbation and post-perturbation examples. For the database perturbations, they aim to simulate

the scenario where data can be represented in different ways in a database. Some examples include the column named country can be replaced with nation, the column named ranking points can be substituted with rank pts. They programmatically modify the database and affected SQLs to generate new data based on sampled perturbations each time. They also simulated the scenario that data can be represented in different formats in a database by replacing a column (column name and content) with one or multiple semantic equivalent columns. For example: For text column content, they consider splitting a compound column into multiple columns, such as replacing full-name with firstname and lastname. They convert columns with fixed string values into boolean column(s) or vice versa. For instance, a boolean column is male is equivalent to a text column sex with all gender options, and a text column degree is equivalent to three boolean columns is bachelor, is master, and is PhD when there are only three degrees in the database. Finally, for numerical columns, they replaced them with other numerical columns with equivalent semantics. For example, the column birthyear is semantically equivalent to the column age since one can be inferred from the other. They also claim that based on the research they did, they are the first to consider representing database content in different ways to create database perturbations.

For the natural language question perturbations, they were collected in a setting that users are familiar with SQL and have the access to the database schema and content. However, in reality, the general users are not familiar with databases or SQLs in most applications. They provided a short description of each database, which reveals the topic of the database and the key tables and columns in the database. Results for the natural language question perturbations ended up being split by filtered data into small chunks and each chunk is reviewed by at least one annotator. 63% generated paraphrases are labeled factual and belong to the correct category, while only 54% of human paraphrases from the previous crowdsourcing are actually factual. This shows the effectiveness of our proposed framework for generating categorized paraphrases.

Finally the SQL perturbations were used to evaluate models' robustness against local semantic changes. They considered five perturbations regarding SQL tokens: (1) Comparison replaces a comparison operation from $\{<, >, \leq, \geq\}$ to another; (2) Sort-order switches ascending sort and descending sort; (3) Non database-number replaces a non-database number n with another number in $[n - 10, n + 10]$, including the number in the LIMIT clause (e.g. Limit n) and a criteria about counts (e.g. HAVING count(*) less than n); (4) Database-text replaces a mentioned DB content text to another in the same column; (5) DB-number replaces a mentioned database content number m to another number in $[m - 10, m + 10]$. Similar to database and NLQ perturbations, we perturb an example at most five times to increase the diversity of perturbations.

Finally looking at the students conclusion, they say “Our experiments reveal existing models are vulnerable to perturbations, while some model architecture designs are more robust against certain perturbations. We envision our findings could benefit future model design for robust text-to-SQL models. In the future, we want to explore different strategies of prompt examples selection besides handpicking for paraphrasing NLQ. We also plan to improve the robustness of text-to-SQL models with the insight from this work, such as combining top-down and bottom-up decoder and using our perturbation methods for data augmentation.”

Something I found very interesting about this paper is that they had a little para-

graph on ethics as well at the end of their research paper. They go on to write: We acknowledge the importance of the ICLR Code of Ethics and agree with it. An insecure text-to-SQL system may cause harm by misinforming its users. Our work aims at addressing it by providing a diagnostic robustness evaluation platform and insights to improve the robustness of models. We construct our benchmark Dr.Spider based on Spider, an public and free text-to-SQL benchmark.

6.2 Reproduction Findings

The goal of natural-language interfaces is to translate queries into executable SQL. With present day technology, large language models have significantly improved. Their improvement also means significant improvement to Text-to-SQL accuracy, however there is no such thing as perfection. With certain changes and shifts in data or languages, the accuracy tends to go down. To make a report on this, students created Dr.Spider. The purpose of Dr.Spider is to diagnose the robustness of Text-to-SQL models. What this means is, it gains in a numerical value the model's ability to consistently generate accurate and executable SQL queries even when faced with variations such as natural language, database schemas, or query structures. The purpose of this recreation is to replicate as many results as possible from the Dr.Spider study on a smaller scale, which is more project and student friendly. Due to issues with the replication, I was not able to fully recreate some results so for those not recreated, I will be doing a more analytical recreation of results. The goal of this recreation is to demonstrate the understanding of the benchmark, the evaluation pipeline and the challenges involved in reproducing these results.

The main question I was trying to solve in doing this replication was can these evaluations using Dr.Spider be replicated using the available GitHub resources. One of the students from the original paper made a GitHub repository with a lot of the main pieces they used for their paper. Their repository can be found here: <https://github.com/awslabs/diagnostic-robustness-text-to-sql?tab=readme-ov-file>. This repository contains all the same data they used as well as a couple of preprocesses and some testing code. The first step for full replication would require the download of these files which I proceeded to do. Once downloaded I was required to make all the folders necessary in the preprocess code so the tables and perturbations were able to be stored accordingly. The experiment required 17 total folders divided amongst the 3 different types of perturbations, natural language, SQL and database. Here is the following data breakups of those 3 perturbations:

Database Perturbations:

- Content equivalence
- Schema abbreviation
- Schema synonym

Natural Language Perturbations:

- Column attribute

- Column carrier
- Column synonym
- Column value
- Keyword carrier
- Keyword synonym
- Multitype
- Others
- Value synonym

SQL Perturbations:

- Comparison
- Database number
- Database text
- Non database number
- Sort order

After debugging, the preprocess code finally finished and all these folders were populated with the correct perturbations and tables, identical to those of the original paper. The next step to the recreation was picking a specific model to test. In the original paper, they ran it through multiple different models such as pre-trained RAT-SQL, LLM prompting and PICARD constrained decoding. I decided to pick a LLM prompting model, more specifically OpenAI, which is GPT-style. My reasoning behind choosing this model was because it's very straightforward: create an account and get the API key that you need to run the code for the experiment. An API key is a code used to authenticate a project or application when it requests access to an application programming interface. OpenAI has no training required and a very clean basic interface to navigate through, so convenience also played a big part in my model choice. Once I got this API key, the next step was to code a program that would use this API key to generate SQL predictions for every perturbation. These predictions were going to be used for each Dr.Spider perturbed question, then when run against the test data, you would receive 2 values, those 2 values being whether it was an exact match and what the execution accuracy was of the model. Then once you have received those 2 values you can move on to calculate the robustness drop of that specific model. You would calculate this by using the original score of the prediction and subtract the score perturbed from it. This is however as far as I made it in my recreation of the experiment, and I will discuss this why a little farther down when I discuss the challenges of the experiment.

For the rest of the recreation will take on more of a conceptual analysis. I will be making these conclusions based on the paper and my own work done. First looking at the database perturbations, based on the paper and my own research, for the abbreviation perturbations, I know that there would be a moderate to small drop because the

values of the database stay the same, abbreviating the column or rows will not change the values. The database schema synonym perturbations would generate a medium to high drop in effectiveness because models like OpenAI rely heavily on these column forms and if this information was to change, there would be a higher drop in the robustness. Looking at the natural language perturbations, I know that the value synonym perturbations would have a small robustness drop because models rely heavily on keywords to run queries. Leading into the next conclusion which is the keyword synonym and keyword carrier perturbations would have a medium to high drop because of the model's reliance on these key words. If the keywords were to change multiple times, heavy drops should be expected. These heavy drops would also be shared with the perturbations column value, column synonym and column carrier as these are important values in a model's ability to run queries. I know the drop would be more moderate than higher as keyword values are even more crucial for a model's performance. Finally looking at the SQL perturbations, the assumption can be made that perturbations database text and number would cause a medium to heavy drop as they are important pieces of information for SQL queries whereas a perturbation like the sort order would be less significant for the model's performance, therefore having a smaller drop in robustness.

Looking at the challenges for myself in this experiment, mainly comes down to the prediction's coding and computer performance. The biggest obstacle was coding a predictions file that works perfectly in line with the Dr.Spider code. No matter how much debugging was done I could not get it to match up with the pre-downloaded code and therefore was unable to run the actual performance script. This was because of the complexity of the Dr.Spider scripts and seeing functions that I have had no experience with. Another huge issue was my computer was struggling to run even the preprocessing script that I previously mentioned. Although fully debugged, this script took multiple minutes to run and complete on my computer and it is because the execution of the script was based on hundreds of databases that were found in the GitHub repository. Despite these complications I was still able to understand the purpose of Dr.Spider and how the pipeline between it and the model works.

In conclusion, although some results were not replicated all the way through, I still feel like I gained a deep understanding of Text-to-SQL queries and the pipeline on how they get translated in LLMs. If I was to try and replicate this experiment again I would use my own dataset to understand everything with the data as well as scale it down to make it more functional for my computer. I would even consider using a different model like PICARD over OpenAI to try and get a different scope on the effectiveness and handling of queries with different models. Even without full reproduction, this project demonstrates that engaging with the expected outcomes and robustness calculations provides meaningful insights into the current state of Text-to-SQL research.

6.3 Idea Extension

There are multiple ways the ideas in this paper can be extended. After doing research it turns out that there is a Dr.Spider 2.0 that works with the new benchmark Spider 2.0. Using the perturbations in a new benchmark is definitely a way to extend the ideas of the paper, and it doesn't necessarily have to be with just Spider 2.0 but it could also use other

benchmarks like BIRD, which is a large text-to-SQL. Using a different data set will also extend the ideas of the paper to see if there is a certain type of data set which is more robust than others. A different data set could even be a very large mess of data to see how well models can perform with for example messy data or grammatically incorrect data. With this mess of data, the models will really be tested on their effectiveness because of the amount of time it would take to find what is required. Those are just a few examples of how the ideas of this paper and experiments be built off of to test the robustness of Text-to-SQL models.

7 Paper 5: Natural SQL: Making SQL Easier to Infer from Natural Language Specifications

7.1 Literature Review

The paper I chose to review was "Natural SQL: Making SQL Easier to Infer from Natural Language Specifications". The NatSQL paper introduces a simplified intermediate representation (IR) of SQL designed to narrow the gap between how people naturally pose questions and how SQL queries must be written. The authors point out that many SQL features, such as GROUP BY, HAVING, explicit JOIN ON clauses, nested sub-queries, and set operators, do not naturally map to the way questions are expressed in everyday language. This mismatch often leads text-to-SQL models to produce incorrect queries. NatSQL streamlines SQL while keeping the essential semantics. It does this by removing several SQL-specific keywords and structures, avoiding unnecessary nesting and set operators, and simplifying schema references. The authors show that training models to produce NatSQL instead of full SQL makes them more accurate and more likely to generate executable queries. On the Spider benchmark, models using NatSQL outperform their standard SQL counterparts and achieve state-of-the-art execution accuracy across multiple baselines.

The main technical contribution of the paper is the NatSQL format itself, along with the conversion rules between NatSQL and standard SQL. The authors define a set of transformations that simplify SQL syntax, add consistent canonicalization for schema elements, and provide post-processing steps to convert model-generated NatSQL back into executable SQL.

To bridge the language gap, NatSQL makes several targeted changes: It removes the need for explicit JOIN ON clauses. Instead, joining tables are implicitly inferred based on foreign key constraints in the database schema, simplifying multi-table queries. Complex SQL features like GROUP BY and HAVING clauses, which are often difficult for models to generate correctly from natural language, are removed. The semantics are captured by simplifying the structure and relying on canonical ordering. NatSQL avoids generating complex, nested subqueries and set operators (UNION, INTERSECT, etc.), which typically do not map directly to simple natural language questions. Instead, it aims for a flatter, more linear query structure. To test its effectiveness, NatSQL is incorporated into several existing text-to-SQL models by replacing their SQL output space with the NatSQL representation. The models are then retrained or fine-tuned to generate NatSQL queries. Evaluation is performed on the Spider dataset, comparing both

exact-match scores and execution accuracy. The paper also includes ablation studies showing which simplifications contribute most to performance improvements.

NatSQL consistently improves model performance across all tested architectures. The paper reports noticeable gains in both syntactic correctness and execution accuracy when models predict NatSQL instead of standard SQL. Importantly, some models that previously struggled to produce executable SQL were able to do so reliably when using NatSQL paired with post-processing to recover full SQL syntax. Across Spider, the NatSQL-based versions of existing models outperform their original forms, demonstrating the benefits of a representation that more closely reflects natural-language phrasing.

Critical Analysis

Strengths:

Highly practical contribution. The paper addresses a commonly observed issue, which is the structural mismatch between natural language and SQL. NatSQL directly targets this problem and shows clear, consistent gains across multiple model types. Model-agnostic design. Because NatSQL is an IR, it can be used with many different modelling approaches, sequence-to-sequence models, constrained decoders, or even large language models, without requiring major architectural changes.

Weaknesses and Limitations:

Potential loss of expressiveness. By removing certain SQL constructs and avoiding nested queries, NatSQL may struggle to represent more advanced or analytical queries that appear in real-world systems but fall outside Spider’s scope. Converting these cases back into full SQL can become complex and cause errors. Reliance on accurate conversion rules. The effectiveness of NatSQL depends heavily on deterministic, reliable SQL to NatSQL mapping. Any gaps or ambiguities in these rules could reduce performance, and implementing the conversion tools correctly can take substantial engineering effort.

Assumptions:

The paper assumes that most natural-language questions correspond well to the simplified subset of SQL that NatSQL preserves. This works for Spider but may not hold for specialized domains or enterprise-level analytical workloads. It also assumes that the NatSQL-to-SQL back-translation is essentially lossless.

Overall, the NatSQL paper provides a compelling argument that simplifying SQL structure can improve model alignment with natural language, though its benefits depend on model familiarity with the representation.

7.2 Reproduction Plan

To reproduce the core claims of the NatSQL paper, we evaluate whether a modern large language model (LLM) can generate NatSQL more accurately than standard SQL in a zero-shot setting. Unlike the original paper, which retrains models to output NatSQL, our reproduction uses an off-the-shelf LLM without fine-tuning.

Our reproduction consists of the following steps:

1. Select a set of 250 questions sampled from the Spider development set, ensuring

coverage of joins, aggregations, nested subqueries, set operators, and boolean logic.

2. For each question, record the gold SQL query provided in the Spider dataset and manually construct gold NatSQL equivalents following the rules in the NatSQL paper.
3. Prompt the LLM with a strict SQL generation instruction:
“Convert the following natural-language question into SQL using correct syntax. Output SQL only.”
4. Prompt the LLM again using a strict NatSQL prompt:
“Convert the following natural-language question into NatSQL following all NatSQL rules. Output NatSQL only.”
5. For both SQL and NatSQL, evaluate syntactic correctness, semantic correctness, and logical equivalence with the gold query.
6. Categorize errors using the taxonomy from the NatSQL paper:
 - join-condition errors
 - grouping logic errors
 - nested subquery failures
 - set-operator errors
 - schema-linking and hallucination errors

7.3 Reproduction Findings

Across all 250 questions, standard SQL significantly outperformed NatSQL in the zero-shot setting. The LLM generated 211 correct SQL queries (84.4%) compared to only 132 correct NatSQL queries (52.8%). This is the opposite of the trend reported in the original NatSQL paper, where models trained on NatSQL achieve higher accuracy.

Table 1 summarizes the aggregate results. Figures 3a and 3b show representative examples of the model’s SQL and NatSQL outputs.

Table 1: Overall correctness of SQL vs. NatSQL across 250 Spider questions.

	SQL	NatSQL
Correct Queries	211	132
Incorrect Queries	39	118
Accuracy (%)	84.4%	52.8%

As shown in Table 1, SQL achieves substantially higher accuracy than NatSQL across all 250 evaluated questions. This outcome contrasts with the findings reported in the original NatSQL paper, where models trained directly on the NatSQL representation demonstrated performance gains. In our reproduction setting using a zero-shot LLM, however, the model consistently performed better with standard SQL. This discrepancy

highlights the extent to which modern LLMs rely on traditional SQL patterns that appear frequently in their pretraining data, whereas NatSQL structures are comparatively rare.

To better illustrate these differences, the following discussion provides detailed qualitative analysis of representative model outputs. We examine common classes of errors, compare structural patterns between SQL and NatSQL generations, and highlight examples that reveal how the absence of explicit JOIN structures or GROUP BY clauses affects the model’s reasoning. The figures following this paragraph present two representative outputs—one for SQL, one for NatSQL—that capture the primary failure patterns observed throughout the full evaluation set of 250 queries.

Question	Gold SQL	NatSQL	Question #	SQL Correct?	NatSQL Correct?	Notes
Show each singer's name and the name of the concert they performed in.	SELECT singer.Name, concert.Name FROM singer JOIN concert ON singer.Concert_ID = concert.Concert_ID;	singer.name, concert.name from singer, concert via Concert_ID	Q1	✗	✗	SQL join ON error; NatSQL simplifies join
List the names of singers and the stadiums where they performed.	SELECT singer.Name, stadium.Name FROM singer JOIN concert ON singer.Concert_ID = concert.Concert_ID JOIN stadium ON concert.Stadium_ID = stadium.Stadium_ID;	singer.name, stadium.name from singer, concert, stadium via Concert_ID, Stadium_ID	Q2	✗	✗	Both failed; multi-table joins + schema unclear caused hallucinations
How many singers are there in each country with more than 1 singer?	SELECT COUNT(*) FROM singer GROUP BY Country; HAVING COUNT(*) > 1;	country, count(*) from singer	Q3	✓	✗	NatSQL incorrectly used GROUP BY/HAVING (not allowed)
What is the average, minimum, and maximum age of all singers from France?	SELECT AVG(Age), MIN(Age), MAX(Age) FROM singer WHERE Country = "France";	avg(Age), min(Age), max(Age) from singer where country = "France"	Q4	✓	✓	Simple aggregation; both succeed
Find the names of students who scored above the average score.	SELECT name FROM student WHERE score > (SELECT AVG(score) FROM student);	name from student where score > avg(score)	Q5	✓	✗	Nested subquery removed in NatSQL
Find professors who teach more courses than the average number taught by all professors.	SELECT name FROM professor WHERE (SELECT COUNT(*) FROM course WHERE course.professor_id = SELECT COUNT(*) AS prof FROM professor GROUP BY prof_id);	name from professor where (SELECT COUNT(*) AS course WHERE course.prof_id = professor_id) > avg(course group by prof_id))	Q6	✗	✗	Very complex; both fail (same as paper)
List plane IDs used for flights but not present in the plane table.	SELECT plane_id FROM flight EXCEPT SELECT id FROM plane;	flight_plane_id from flight, plane via plane_id where plane_id is null	Q7	✗	✗	EXCEPT is simplified in NatSQL
Find the cities that appear both as departure and arrival cities.	SELECT departure_city FROM flight INTERSECT SELECT arrival_city FROM flight;	departure_city from flight	Q8	✓	✗	SQL correct; NatSQL invalid due to aliases and DISTINCT
Find properties that are houses or apartments.	SELECT * FROM property WHERE type = "house" OR type = "apartment";	* from property where type = "house" or type = "apartment"	Q9	✓	✓	Simple OR filter
Find houses or apartments with more than 2 rooms.	SELECT * FROM property WHERE type = "house" OR type = "apartment" AND rooms > 2;	* from property where type = "house" or type = "apartment" and rooms > 2	Q10	✗	✗	Simple boolean logic

(a) SQL vs. NatSQL comparison

(b) NatSQL typical structural errors

Figure 3: Comparison of results and typical errors for NatSQL.

Error Patterns. SQL errors typically involved ambiguous joins, incorrect nested logic, or occasional hallucinated table names. By contrast, NatSQL errors were more severe and more frequent:

- misuse or omission of “via” join notation,
- reintroducing disallowed SQL constructs (GROUP BY, HAVING, nested SELECT),
- incorrect flattening of nested semantics,
- set operator misuse,
- schema inference failures.

These findings suggest that NatSQL’s simplifications do *not* help a zero-shot LLM. Instead, they remove structural cues that the model relies on when inferring relationships between tables. Because LLMs are heavily trained on standard SQL but rarely exposed to NatSQL, they handle SQL far more reliably.

7.4 Idea Extension

Our results indicate that NatSQL is unlikely to benefit standard zero-shot prompting. However, the representation could still be valuable if integrated into an LLM pipeline with targeted supervision. We propose a lightweight two-stage extension:

1. Few-shot NatSQL prompting. Provide the LLM with a set of 20–30 high-quality demonstration pairs (natural language → NatSQL). This would give the model direct exposure to NatSQL syntax.
2. Automated post-processing. Use the deterministic SQL/NatSQL conversion scripts from the original paper to translate LLM-generated NatSQL into executable SQL.

This hybrid approach preserves NatSQL’s structural simplicity while avoiding full model retraining. It aligns closely with practical use cases where developers want better SQL generation without the cost of training a full text-to-SQL system. Future work could also evaluate chain-of-thought join inference, schema linking using retrieval, or fine-tuned adapters to better support NatSQL’s flattened representation.

8 Conclusion

Throughout all 5 papers and reproductions examined, several key themes emerged regarding current progress and limitations in Text-to-SQL research. Structured prompting approaches such as SQLPrompt showed performance improvements, even in minimal-data settings. Benchmark and robustness-focused datasets like DR.Spider highlighted how model performance differs with changes in data, showing how when an environment isn’t controlled, models tend to struggle. While modern LLMs achieve strong results on foundational datasets like Spider 1.0, the newer Spider 2.0 benchmark shows major differences: execution accuracy remains low, methods currently outperform standard approaches, and robust deployable systems are still lackluster. The literature also shows that some amateur representations, such as NatSQL, does not consistently provide benefits in certain scenarios without external help. At the same time, optimization frameworks such as DAIL-SQL show how systematic tuning pipelines can get accuracy higher and broaden our understanding of model behavior. Together, these findings emphasize the importance of clear SQL structure, careful evaluation design, and concise benchmarking for advancing Text-to-SQL reliability.

References

- B. Li *et al.*, “NatSQL: Making SQL Easier to Infer from Natural Language Specifications,” in *Findings of ACL/IJCNLP*, 2021.
- D. Gao *et al.*, “Text-To-SQL Empowered by Large Language Models: A Benchmark Evaluation,” *Proc. VLDB Endowment*, vol. 17, no. 5, pp. 1132–1145, 2024.
- L. Shi *et al.*, “A Survey on Employing Large Language Models for Text-to-SQL Tasks,” *ACM Computing Surveys*, 2025.

R. Sun *et al.*, “SQLPrompt: In-Context Text-to-SQL with Minimal Labeled Data,” in *Findings of EMNLP*, 2023.

T. Yu *et al.*, “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task,” in *Proc. EMNLP*, 2018.

Toloka Team, “The History, Timeline, and Future of LLMs,” blog article, 2023. [Online]. Available: <https://toloka.ai/blog/history-of-l1ms/>