

# Group 3 Final Project Report

## COSC 5600

Noah Nieberle  
Marquette University  
[noah.nieberle@marquette.edu](mailto:noah.nieberle@marquette.edu)

Harrison Salmon  
Marquette University  
[harrison.salmon@marquette.edu](mailto:harrison.salmon@marquette.edu)

Shaun Campbell  
Marquette University  
[shaun.campbell@marquette.edu](mailto:shaun.campbell@marquette.edu)

Elizabeth Meyer  
Marquette University  
[e.g.meyer@marquette.edu](mailto:e.g.meyer@marquette.edu)

## 1. Introduction

In the digital era, vast amounts of data are stored in relational databases. Structured Query Language, or SQL, is a powerful programming tool that allows for the querying and retrieving of data from relational databases. However, writing SQL often requires specialized knowledge in correct SQL syntax and programming practices, and this can be an obstacle for non-technical users who may also wish to use SQL to retrieve and manipulate data [6].

Text-to-SQL is a topic in the field of natural language processing (NLP) that involves converting natural language into SQL queries. An example would be the natural language prompt: “Find the names of all students with a GPA of 3.5 or higher.” A Text-to-SQL algorithm would then ideally generate the correct SQL query from the natural language prompt, which would be: “SELECT student\_name FROM students WHERE GPA  $\geq$  3.5”.

As the field of deep learning has advanced, Large Language Models (LLMs) have emerged as a new approach that shows great promise in Text-to-SQL tasks. The sheer amount of data and emergent capabilities unique to LLMs have allowed them to soar past other traditional Text-to-SQL methods. LLMs are unique in that they have two main approaches for Text-to-SQL applications: prompt engineering, which takes advantage of the instruction-following capabilities of a LLM and its ability to employ reasoning techniques, and fine-tuning methods, which involve training a pretrained LLM on text-to-SQL datasets for additional context and learning material [6].

To demonstrate LLM capabilities on Text-to-SQL tasks, we applied the DIN-SQL methodology to the Spider [11] and BIRD Mini Dev [3] benchmark datasets, using Gemini 2.5 Flash as the LLM. Performance was measured using the Execution Accuracy (EX) metric.

## 2. Related Work

### 2.1. Template-Based Approaches

The Text-to-SQL process is a highly active research challenge that has had many varying approaches applied in its fifty year lineage. The pioneering approaches from the early 1970s introduced the concept of “rule-driven semantic interpretation” [9], where a question asked in natural language would be run through a set of rules to interpret it. This process was laborious and struggled to interpret complex ideas, however laid the groundwork for the development of more sophisticated tools.

### 2.2. Recurrent Neural Networks

Capitalizing on natural language’s sequential nature, Recurrent Neural Networks (RNNs) were a framework that held great promise to parse natural language well enough to produce effective SQL queries. RNNs iteratively build up a fixed size hidden state that reflects the content of the sequence. The benefit of RNNs is the ability to retain important information revealed at the beginning of a sequence by only dropping information that is not critical. Long Short Term Memory [2] was a landmark paper that explored this strategy and methods such as SQLNet [10] have implemented this approach in a Text-to-SQL landscape. Despite a clear leap in complexity, RNNs had technical roadblocks of their own, such as the vanishing/exploding gradient problem and long processing times in training due to the sequential nature of the data. Large Language Models (LLMs) have since made another leap forward in this space.

### 2.3. Large Language Models

Widely accessible LLMs have demonstrated a strength in performing natural language semantic interpretation as well as being able to generalize. As a result, the development of custom built deep learning models to convert natural

language to SQL has been outsourced to many different open and closed sourced LLMs. There are two predominant methods the Text-to-SQL community have focused their research on in this new paradigm, fine-tuning and prompt engineering.

### 2.3.1. Fine-Tuning

Fine-tuning involves using open sourced models such as DeepSeek, Llama and Qwen to make customizations that improve their performance in the specific domain of improving Text-to-SQL problems. Using this approach sidesteps data privacy issues associated with sharing sensitive information with closed source LLMs; however large hardware requirements can be a significant barrier to entry into this area of research. Finely tuned open source models, such as Code Llama have options for the training of up to 70b parameters [5]. Despite this investment, a model such as this will still be dwarfed by much larger closed source models such as the 1.8 trillion (1800 billion) parameter GPT-4. The benefits of Text-to-SQL fine tuning must outweigh the less focused yet highly trained and adaptability of closed source models.

### 2.3.2. Prompt Engineering

Prompt engineering leverages the computational power of closed source LLMs and combines it with specific instructions and context clues to increase the probability of accurate responses. Including details on the database schema and additional domain context in the prompt can assist the LLM in overcoming issues with understanding domain specific language and a lack of knowledge of the database structure.

Minimising the expense of using LLMs is a restriction that can inhibit the effectiveness of modern Text-to-SQL software. A common method of improving the generated SQL query is to make recursive calls to a LLM to iteratively make improvements. The fact that many LLMs are operated as a paid service or have call restrictions can limit the effectiveness of many techniques. Approaches such as CHESS [7] provide a base strategy for the users comfortable using many API calls while having provisions that adjust the software in order to maximise the value of a limited number of calls for users with that restriction.

## 3. Methodology

Decomposed In-Context Learning of Text-to-SQL with Self-Correction (DIN-SQL) [4] is a methodology to generate a SQL query from a natural language prompt via an LLM. It is a few-shot approach, meaning that a minimal number of examples, or in this case demonstrations, are given to the model. The model can then generalize from the demonstrations and apply the learned response patterns to a new problem.

The DIN-SQL method uses the chain-of-thought approach [8] to construct prompts. This approach aims to assist the LLM in reasoning in a step-by-step manner by guiding it to follow a series of intermediate steps. One simple way in which chain-of-thought can be implemented is by including language such as “Let’s think step by step” in the prompt.

The DIN-SQL method decomposes the text-to-SQL problem into four subtasks: schema linking, classification and decomposition, SQL generation, and self-correction.

### 3.1. Schema Linking

The schema linking task is designed to help the LLM identify relevant aspects of the database schema, such as tables, foreign keys, and cell values. It also introduces the schema of the database into the context of the LLM. For each table in the database, the LLM is given the table names and column names in the format:

Table *table name*, columns = [*column names*]

The table relationships are given in the following format, where pk is the table’s primary key and fk is the related table’s foreign key:

Foreign\_keys = [*table1.pk = table2.fk, table2.pk = table3.fk, ...*]

Ten pre-written demonstrations of the task performed on example databases are included first in the prompt, then the schema for the database to which the question pertains to is given. The natural language question is asked, and the LLM is told “Let’s think step by step”. For example, for a question relating to a soccer database, “Show names of soccer players and the name of the club they are in”, the generated schema links should be [player.Name, club.Name, player.Club\_ID = club.Club\_ID]. The schema links are then parsed from the response and saved for later use.

### 3.2. Classification and Decomposition

The classification and decomposition task aims to identify the difficulty of the question depending on whether joins and subqueries are needed. If no joins or subqueries are needed, the LLM should classify the question as “easy”. If join(s) are needed but subqueries are not, the question should be classified as “non-nested”. Finally, if both joins and subqueries are needed, the question should be classified as “nested”. The classification of the question determines which SQL generation prompt method is used in the next task.

In addition to labels, this task also asks the LLM to identify the tables that are needed, and any sub-questions for nested queries. The required tables are not parsed from the

response, but having the LLM identify them aims to introduce them into the model’s context. Any sub-questions that are generated are parsed out, and are provided to the LLM again in the SQL generation step.

The natural language questions, as well as the schema links that were identified in the previous step, are provided to the LLM in the prompt. 10 pre-written demonstrations of the task performed on example databases are also included first in the prompt. For the previous soccer database question, the LLM should classify it as “non-nested” because a join (`player.Club_ID = club.Club_ID`) is required.

### 3.3. SQL Generation

The goal of this task is to have the LLM generate an executable SQL query that answers the natural language prompt. Different prompting methods are used based on the difficulty classification from the previous step.

For easy questions, just the natural language question and schema links are provided. This is based on research that has shown more complex chain-of-thought prompting can degrade performance on simple text-to-SQL tasks [8].

For non-nested questions, the natural language question and schema links are provided as well. The LLM is also guided, through the demonstrations that are provided, to generate an intermediate representation of the query to assist it in reasoning towards the correct answer. The intermediate representation used in DIN-SQL is based on a method called NatSQL [1]. This method removes or simplifies the operators `JOIN ON`, `FROM`, `GROUP BY`, `UNION`, `UNION ALL`, `INTERSECT`, and `EXCEPT`.

Finally, for nested questions, the natural language question and schema links are provided, and the LLM is guided to form an intermediate representation. Additionally, the LLM is given the sub-question(s) identified in the previous step. The LLM is guided, through demonstrations, to first form SQL queries to answer the sub-question(s) then compile those into the final query.

In our implementation, we also provided the LLM with three sample rows of each table of the relevant database in the SQL generation prompt. The authors of DIN-SQL note they did this for their tests on BIRD, but not on Spider. The authors used different prompts for BIRD which included example rows for the demonstration databases. We used the same prompts for both Spider and BIRD, and did not include example rows for the demonstrations, only example rows for the question-specific database.

### 3.4. Self-Correction

This task aims to correct minor syntactic errors in the generated SQL query to improve the chances that the query will execute. No demonstrations are given to the LLM, just the SQL query that was generated in the previous step. A “gentle” approach is used, where the LLM is not told explicitly

that the SQL is buggy, but merely asked to check the SQL and correct issues if any are found.

## 4. Experimental Evaluation

The experimental evaluation was designed to rigorously measure the performance of the implemented DIN-SQL system, which utilizes Google’s Gemini API to translate natural language questions into SQL queries. The evaluation focused on two complementary datasets: the Spider benchmark [11], which represents well-structured academic relational databases, and the BIRD Mini Dev benchmark [3], which introduces more realistic, domain-intensive schemas. These datasets were chosen deliberately to test the system’s generalizability across environments of differing complexity. For both benchmarks, the pipeline consisted of constructing prompts enriched with schema context, generating SQL queries via the model, executing the queries against the corresponding databases, and finally computing execution accuracy, which reflects whether the predicted SQL yields the same results as the gold standard. Because execution accuracy captures both syntactic and semantic correctness, it is regarded as one of the most meaningful and stringent metrics in text-to-SQL evaluation, making it appropriate for this study.

### 4.1. Spider

On the Spider benchmark, the system achieved an execution accuracy of 83.47%, which is competitive with recent prompt-based approaches and demonstrates that the model reliably interprets schema structures and produces meaningful SQL across a diverse set of academic domains. Most databases in the Spider dataset achieved above 90% accuracy, indicating that the system is particularly effective when schemas are moderately sized and attribute names follow conventional academic naming conventions. The remaining errors concentrated in a few challenging databases with dense relational structures or ambiguous attribute names. Databases such as `art_1`, `real_estate_rentals`, and `address_1` contain many tables with similarly named attributes and non-obvious join paths, which are known to introduce difficulties for schema linking. The model’s struggles in these cases align with challenges documented in existing literature, suggesting that while the current prompting and decomposition strategy is robust for the majority of Spider queries, there is still sensitivity to schema ambiguity and complex join reasoning.

### 4.2. BIRD Mini Dev

In contrast, performance on the BIRD Mini Dev benchmark decreased substantially, with the system achieving 43.2% execution accuracy. This drop is expected and widely reflected in prior work because BIRD introduces many of the challenges absent in Spider, including large schemas,

domain-specific fields, and intricate, paraphrased natural language questions. Errors varied dramatically across individual databases. Domains such as toxicology, clinical thrombosis prediction, and financial regulation exhibited the weakest performance, illustrating the model’s difficulty in handling specialized terminology and multi-step reasoning inherent in these fields. Simpler domains performed noticeably better, but the dataset overall remained difficult due to the variety and complexity of the question types. The difficulty breakdown further corroborated this pattern: accuracy on simple queries reached 61%, but moderate and challenging queries dropped to 38% and 28%, respectively. This steep gradient underscores the model’s limitations in performing multi-hop logical reasoning, constructing nested SQL queries, and inferring implicit domain rules solely from schema text.

Taken together, these results paint a clear picture of the system’s strengths and weaknesses. The high performance on Spider shows that the DIN-SQL prompting strategy generalizes well to traditional academic database environments, where schemas are clean and linguistic patterns are familiar. However, the lower performance on BIRD reveals that the system is less capable of managing real-world schema complexity, domain-heavy terminology, and queries requiring sophisticated, multi-stage reasoning. These findings suggest several promising directions for improvement, including richer schema representations in prompts, more explicit reasoning decomposition steps, and iterative self-correction mechanisms capable of identifying logical inconsistencies before execution. Overall, the evaluation confirms that the system is highly effective within typical benchmark conditions while also highlighting areas that must be strengthened to achieve robust, real-world performance.

## 5. Division of Work

At the start of the project, all group members independently researched related papers and conducted initial experimentation with DIN-SQL and other text-to-SQL approaches. After jointly determining that DIN-SQL was the most effective method, each member attempted partial implementations to build a shared understanding of the system before dividing responsibilities for the final report and experiments.

For the final stage of the project, work was divided as follows: Liz Meyer wrote the Introduction & Background and the Conclusion, establishing the report’s framing and summarizing key findings. Harrison Salmon authored the Related Work section, situating DIN-SQL within existing research. Shaun Campbell prepared the Methodology, detailing the implementation pipeline and prompting strategy. Noah Nieberle completed the Experimental Evaluation, running the experiments, generating tables and plots, and analyzing system performance.



Figure 1. BIRD Mini Dev Accuracy by Difficulty (top) and Overall on Spider and BIRD Mini Dev (bottom).

## 6. Conclusion

This project explored the effectiveness of applying LLMs to Text-to-SQL tasks by implementing the DIN-SQL methodology and evaluating it on the Spider and BIRD Mini Dev benchmark datasets using Gemini 2.5 Flash. Our implementation successfully translated natural language prompts into executable SQL queries, achieving strong performance on the Spider dataset (83.47% EX) and more limited execution accuracy on the complex BIRD Mini Dev dataset (43.2% EX). These results demonstrate both the promise of LLM-based approaches for Text-to-SQL applications and the remaining challenges in handling real-world schema complexity and advanced reasoning.

## References

- [1] Y. Gan, X. Chen, J. Xie, M. Purver, J. R. Woodward, J. Drake, and Q. Zhang. Natural SQL: Making SQL easier to infer from natural language specifications. *arXiv preprint arXiv:2109.05153*, 2021. 3
- [2] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 1
- [3] J. Li, B. Hui, G. Qu, B. Li, J. Yang, B. Li, B. Li, B.

- Wang, B. Qin, R. Cao, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. Chang, F. Huang, R. Cheng, and Y. Li. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. *arXiv preprint arXiv:2305.03111*, 2023. [1](#), [3](#)
- [4] M. Pourreza and D. Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. *Advances in Neural Information Processing Systems*, 36: 36339–36348, 2023. [2](#)
- [5] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. [2](#)
- [6] L. Shi, Z. Tang, N. Zhang, X. Zhang, and Z. Yang. A survey on employing large language models for text-to-SQL tasks. *ACM Computing Surveys*, 58(2):1–37, 2025. [1](#)
- [7] S. Talaei, M. Pourreza, Y. Chang, A. Mirhoseini, and A. Saberi. CHESS: Contextual harnessing for efficient SQL synthesis. *arXiv preprint arXiv:2405.16755*, 2024. [2](#)
- [8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022. [2](#), [3](#)
- [9] W. A. Woods, R. M. Kaplan, and B. Nash-Webber. The lunar sciences natural language information system: Final report. 1972. [1](#)
- [10] X. Xu, C. Liu, and D. Song. SQLnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017. [1](#)
- [11] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Raden. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. *arXiv preprint arXiv:1809.08887*, 2018. [1](#), [3](#)