Text-to-SQL Generation using Dynamic Few-Shot Prompting

Choudam Jaitej (251110023)

Shashikanth Kungulwar (251110066)

Vishnu Vardhan Reddy Tavanam (251110076)

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur (IIT Kanpur)

Under the guidance of:

Prof. Arnab Bhattacharya
Prof. Subhajit Roy

November 14, 2025

Contents

1 Introduction

1.1 Background

Text-to-SQL is an important research area within natural language processing (NLP) and artificial intelligence (AI). It focuses on translating natural language questions into structured SQL queries, allowing users to interact with databases without requiring technical expertise. This capability makes data retrieval more intuitive and promotes easier access to information for people from non-technical backgrounds.

The foundation for this work is based on the **MAC-SQL** paper, which served as our primary reference and baseline for understanding the Text-to-SQL problem. The datasets used in this project were taken from the same sources as those used in the MAC-SQL study, ensuring consistency and comparability with previous research. Building upon this base, our work explores the use of dynamic few-shot prompting techniques to further improve the accuracy and validity of SQL generation from natural language queries.

1.2 Problem Statement

Generating accurate and executable SQL queries from natural language remains challenging, particularly when dealing with complex database schemas and questions, as demonstrated by benchmark datasets like Spider and BIRD. This project addresses the need for effective prompting strategies to maximize the performance of Large Language Models (LLMs) in this domain.

1.3 Project Goal

The objective of this project is to investigate and compare the effectiveness of various prompting techniques, focusing on static and dynamic few-shot learning, using the **Gemini** and **Llama** models on the **Spider** and **BIRD** datasets.

2 Methodology

2.1 Experimental Setup

- **LLMs Used:** Three models were used across all Text-to-SQL generation experiments, serving different purposes within the workflow:
  - ◇ **Google Gemini** (`gemini-2.5-flash`) — used as one of the primary large language models for Text-to-SQL query generation.
  - ◇ **Llama** (`llama-3.1-8b-instant`) — also used for Text-to-SQL generation to compare performance and prompting effectiveness.
  - ◇ **all-MiniLM-L6-v2** — a **SentenceTransformer** model employed to generate vector embeddings of natural language questions. These embeddings were used to retrieve the most semantically similar examples for constructing dynamic few-shot prompts.
- **Datasets Used:**The experiments were conducted using the **Spider** and **BIRD** datasets, both of which are widely recognized benchmarks in the Text-to-SQL research community. Due to limitations in available resources such as API access quotas and model usage constraints, only a subset of **200 samples** from each dataset was selected for testing

and evaluation. The remaining samples **834** instances from Spider and **1334** instances from BIRD — were used as the knowledge base for retrieving the top-K most relevant examples during dynamic few-shot prompting.

- **Evaluation Metrics:** The performance of each model was measured using two key metrics:

  ◇ **Execution Accuracy (Ex):** This metric evaluates whether the generated SQL query, when executed on the target database, produces the correct output. It directly measures the functional correctness of the query.

$$\text{Ex} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{R_i^{\text{pred}} = R_i^{\text{gold}}\}$$

  ◇ **Valid Efficiency Score (VES):** This metric assesses how valid and executable a generated query is while also reflecting its overall efficiency and logical soundness. A higher VES indicates that the model not only generated a syntactically correct SQL statement but also produced one that is efficient and semantically appropriate for the given question.

$$\text{time\_ratio}_i = \begin{cases} \dfrac{t_i^{\text{gold}}}{t_i^{\text{pred}}}, & t_i^{\text{pred}} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\text{VES} = \frac{1}{N}\sum_{i=1}^{N} 100\,\sqrt{\text{time\_ratio}_i}$$

2.2 Prompting Techniques

2.2.1 Normal and Structured Prompting

Normal Prompting (Simple Baseline): For the baseline method, we used a very simple zero-shot prompt that only included the database schema and the user's natural language question. This prompt does not enforce any structure or SQL-only output, so it serves as a raw baseline for comparison.

```
Below is an instruction that describes a task, paired with an input that
provides further context.
Write a response that appropriately completes the request.


### db_info:\n{db_schema}\n
### Input:\n{user_question}\n
### Response:
```

In practice, this prompt produced highly variable outputs. While it sometimes generated a correct query for simple questions, the model also frequently included natural language explanations or hallucinated table and column names. Because SQL format was not enforced, this approach required manual cleaning or additional post-processing.

Structured Prompting: To improve consistency and reduce hallucinations, we used a more explicit structured prompt. Here, we specify that the model is an SQL expert, provide the database schema, and enforce a two-part output format: a short reasoning section followed by an SQL query starting with `SELECT` and ending with a semicolon.

```
You are an expert data scientist and SQL engineer.
You are given a database schema and a natural language question.
Your task is to carefully analyze the schema, reason about table
relationships,
and generate a correct and efficient SQLite query that fully answers the
question.


Database Engine:  SQLite


Now, for the following schema and question, follow the same reasoning
process and output format:


Database Schema:  {db_schema}
Question:  {user_question}


- Reasoning (2-6 lines)
- SQL Query (output starts with SELECT and ends with ;)
Return output in the required format and nothing else.
```

This structured prompt significantly improved the quality and formatting of SQL output. The model adhered better to the schema, used correct join paths more frequently, and produced output that was easier to evaluate automatically.

2.2.2 Static Few-Shot Prompting

In the static few-shot approach, we prepend a fixed set of example (Question, Reasoning, SQL) triplets to every prompt. These examples are constant for all test inputs, and they help the model learn the desired format, reasoning structure, and SQL generation patterns. The complete few-shot prompt used in our system is shown below.

```
You are an expert data scientist and SQL engineer.  You are given a
database schema and a natural language question.  Your task is to
carefully analyze the schema, reason about table relationships, and
generate a correct and efficient SQLite query that fully answers the
question.


Database Engine:
SQLite


Follow the reasoning and query structure as shown in the few-shot
examples below.
```

```
Example 1:
- Question
Find the names of all students majoring in 'Computer Science'.
- Reasoning
We need student names and their corresponding major.  Join `students`
with `majors` on `major_id`, then filter for major_name = 'Computer
Science'.
- SQL Query
SELECT s.name
FROM students AS s
JOIN majors AS m ON s.major_id = m.major_id
WHERE m.major_name = 'Computer Science';


Example 2:
- Question
Find the average salary for each department.
- Reasoning
We group employees by department and compute the average salary for each
group.
- SQL Query
SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department;


Example 3:
- Question
List the names of customers who placed more than 3 orders.
- Reasoning
We join customers with orders, group by customer, and count their orders.
Filter those with count > 3.
- SQL Query
SELECT c.customer_name
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
GROUP BY c.customer_name
HAVING COUNT(o.order_id) > 3;


Example 4:
- Question
Find the top 3 highest-rated movies released after 2015.
- Reasoning
We filter movies by release_year > 2015, order them by rating descending,
and limit to 3 results.
- SQL Query
SELECT title, rating
```

```
FROM movies
WHERE release_year > 2015
ORDER BY rating DESC
LIMIT 3;


Example 5:
- Question
Find the total sales revenue (price * quantity) for each product.
- Reasoning
Join products with sales on product_id, multiply price by quantity, then
group by product name to sum the revenue.
- SQL Query
SELECT p.name, SUM(p.price * s.quantity) AS total_revenue
FROM products AS p
JOIN sales AS s ON p.product_id = s.product_id
GROUP BY p.name;


Now, for the following schema and question, follow the same reasoning
process and output format:


Database Schema:  {db_schema}
Question:  {user_question}


- Reasoning (2-6 lines)
- SQL Query (output starts with SELECT and ends with ;)
Write only the SQL query and nothing else.
```

Static few-shot prompting significantly improved formatting consistency and reduced hallucinations. By showing the model concrete examples of typical SQL patterns, it produced more reliable queries. However, because the same examples were used for every question, performance could degrade when the test question did not resemble any of the fixed demonstrations.

2.2.3 Dynamic Few-Shot Prompting

We use a separate knowledge file containing solved question–answer pairs that are not part of the model's training set. For each new input question, we obtain vector embeddings for both the input and each knowledge example using the `all-MiniLM-L6-v2` SentenceTransformer model:

$$e(q) = \text{SentenceTransformer}(q).$$

These embeddings place semantically similar texts closer together in the vector space. To measure relevance, we compute cosine similarity between the embedding of the input question q and each candidate example x_i:

$$s(q, x_i) = \frac{e(q)^\top e(x_i)}{\|e(q)\| \, \|e(x_i)\|}.$$

We rank all candidate examples by their similarity scores and select the top-5 most similar examples. These retrieved examples are then appended to the prompt as few-shot demonstrations for the model.

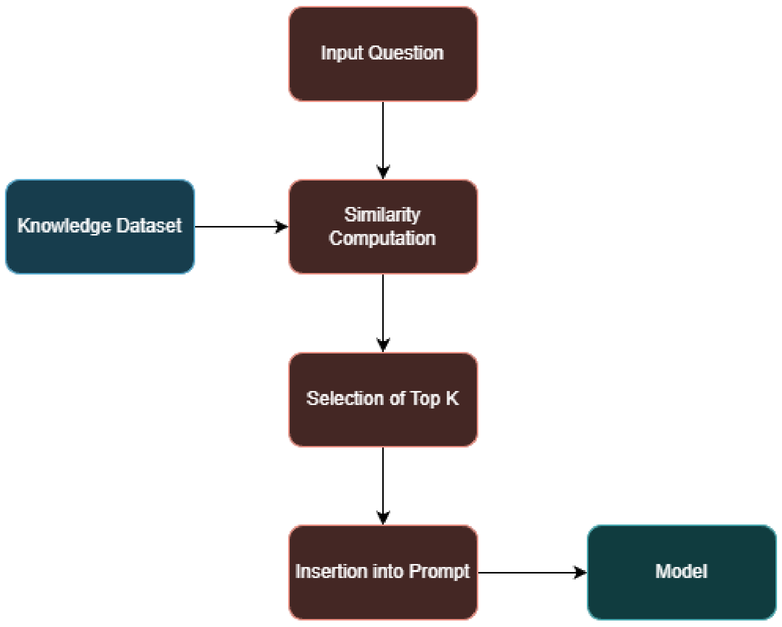

Figure 1: Dynamic Few-Shot Workflow

The retrieved examples are incorporated into the prompt (as {top_k}) to provide contextual guidance. The final prompt structure used for query generation is shown below:

```
You are an expert data scientist and SQL engineer.
You are given a database schema and a natural language question.
Your task is to carefully analyze the schema, reason about table
relationships,
and generate a correct and efficient SQL query that fully answers the
question.
Only output valid SQL (no explanation or English text).


Database Engine:  SQLite


Few-shot examples:  {top_k}


Now do the same for:


Database Schema:  {db_schema}
Question:  {user_question}


- Reasoning (2-6 lines)
- SQL Query (output starts with SELECT and ends with ;)
```

We systematically vary the number of retrieved examples and evaluate configurations with $K = 1, 2, 3, 4, 5$ to study the influence of few-shot context size. The corresponding results and analysis are presented in the subsequent section.

3 Results and Analysis

3.1 Overview

This section presents a comparative analysis of the four prompting strategies explored in this study: Simple Prompting, Structured Prompting, Static Few-Shot Prompting, and Dynamic Few-Shot Prompting. The evaluation was performed on both the **Gemini** and **Llama** models using the **Spider** and **BIRD** datasets. Model performance was assessed using two primary metrics, **Execution Accuracy (Ex)** and **Valid Efficiency Score (VES)**. Execution Accuracy measures how correctly the generated SQL query retrieves the intended result when executed, while the Valid Efficiency Score evaluates the validity, logical soundness, and efficiency of the generated SQL queries. This comparison helps in understanding how different prompting strategies impact model performance across datasets of varying complexity.

3.2 Simple Prompting Results

This baseline method directly maps natural language questions to SQL queries without schema context or examples.

Table 1: Simple Prompting Accuracy Comparison

Model	**Spider**		**BIRD**	
	Ex (%)	Ves (%)	Ex (%)	Ves (%)
Gemini	61.00	60.84	33.00	101.66
Llama	43.50	59.72	8.00	45.76

3.3 Structured Prompting Results

Structured prompting included schema information (DDL) and a standardized output format, leading to improved syntactic correctness.

Table 2: Structured Prompting Accuracy Comparison

Model	**Spider**		**BIRD**	
	Ex (%)	Ves (%)	Ex (%)	Ves (%)
Gemini	71.00	77.51	33.00	99.15
Llama	44.50	59.58	20.00	67.64

3.4 Static Few-Shot Prompting Results

A fixed set of N curated examples were prepended to all test prompts to enable in-context learning.

Table 3: Static Few-Shot Prompting Accuracy Comparison

Model	**Spider**		**BIRD**	
	Ex (%)	Ves (%)	Ex (%)	Ves (%)
Gemini	72.00	69.96	33.00	103.95
Llama	44.50	57.34	17.50	49.63

3.5 Dynamic Few-Shot Prompting Results

Dynamic few-shot prompting dynamically retrieves the top-K most similar examples per question using embeddings and cosine similarity. This section reports performance variations across different K values on both the **Spider** and **BIRD** datasets.

3.5.1 Spider Dataset Results

Table 4: Dynamic Few-Shot Accuracy on Spider Dataset

Model	**K = 1**	**K = 2**	**K = 3**	**K = 4**	**K = 5**
Gemini (Ex %)	71.00	71.50	72.50	72.50	74.50
Gemini (Ves %)	71.55	71.89	74.01	75.69	80.43
Llama (Ex %)	46.00	44.50	50.00	49.00	48.50
Llama (Ves %)	64.25	63.52	63.40	59.61	62.18

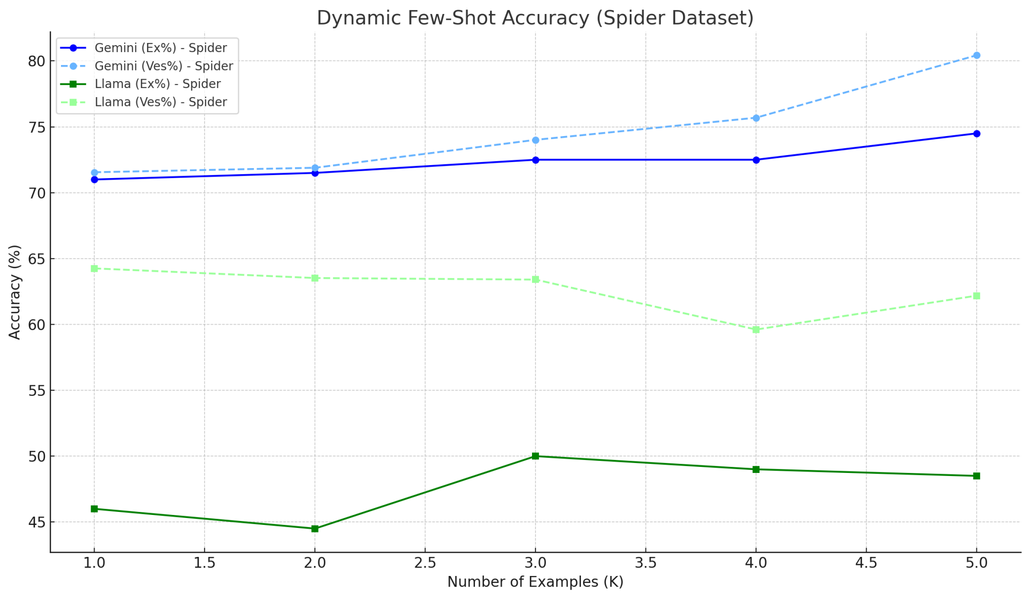

Figure 2: Execution Accuracy (Ex) , (VES) vs. Number of Examples (K) in Dynamic Few-Shot Prompting for Spider

Table 4 presents the dynamic few-shot results on the Spider dataset across different values of K. We observe that Gemini exhibits consistent performance gains as K increases, achieving its highest performance at $K = 5$ (74.50% execution accuracy and 80.43% VES, where VES denotes the Valid Efficiency Score). In contrast, Llama shows only modest variation across different K values, with performance stabilizing around $K = 3$–4. This indicates that Gemini is able to make more effective use of additional contextual examples, whereas Llama benefits only marginally from larger few-shot contexts on this dataset.

3.5.2 BIRD Dataset Results

Table 5: Dynamic Few-Shot Accuracy on BIRD Dataset

Model	**K = 1**	**K = 2**	**K = 3**	**K = 4**	**K = 5**
Gemini (Ex %)	47.50	50.00	50.50	52.00	52.50
Gemini (Ves %)	84.18	90.15	90.28	86.01	84.05
Llama (Ex %)	26.00	29.50	29.50	32.50	32.50
Llama (Ves %)	50.00	40.22	75.53	81.84	82.00

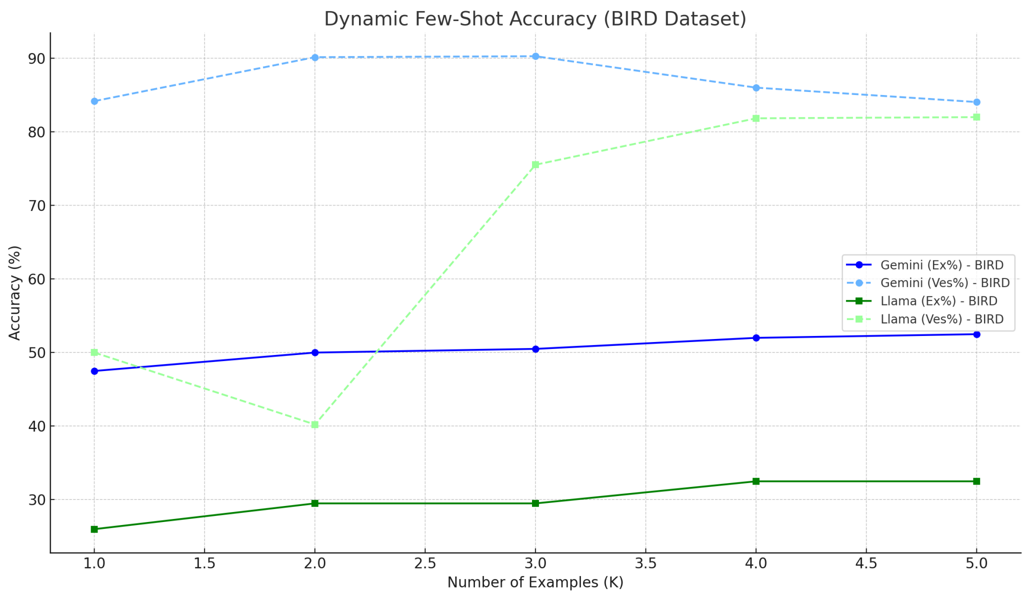

Figure 3: Execution Accuracy (Ex) ,Valid Efficiency Score (VES) vs. Number of Examples (K) in Dynamic Few-Shot Prompting for Bird

Table 5 presents the dynamic few-shot results on the BIRD dataset across different values of K. For Gemini, we observe a moderate improvement in execution accuracy as K increases, with performance stabilizing around K = 3–5. However, its VES remains consistently high across all values of K, indicating that Gemini is able to generate valid and efficient SQL queries even with fewer contextual examples.

For Llama, the execution accuracy improves slightly with higher K, but the VES shows a more noticeable rise, peaking at $K = 4$. This suggests that while Llama initially struggles with execution correctness at low K, providing additional few-shot examples helps the model produce more structurally efficient and semantically valid SQL queries on the BIRD dataset.

3.6 Comparative Summary

Table 6: Performance of **GPT4 + MAC-SQL** on Spider and BIRD

Model	**Spider**		**BIRD**	
	Ex (%)	**VES (%)**	**Ex (%)**	**VES (%)**
GPT4+ MAC-SQL	82.80	–	59.59	67.68

Table 7: Accuracy Comparison Across Prompting Techniques for **Gemini**

Prompting Method	**Spider**		**BIRD**	
	Ex (%)	**VES (%)**	**Ex (%)**	**VES (%)**
Simple Prompt	61.00	60.84	33.00	101.66
Structured Prompt	71.00	77.51	33.00	99.15
Static Few-Shot	72.00	69.96	33.00	103.95
Dynamic Few-Shot (Best K)	74.50	80.43	52.50	90.28

Table 8: Accuracy Comparison Across Prompting Techniques for **Llama**

Prompting Method	**Spider**		**BIRD**	
	Ex (%)	**VES (%)**	**Ex (%)**	**VES (%)**
Simple Prompt	43.50	59.72	8.00	45.76
Structured Prompt	44.50	59.58	20.00	67.64
Static Few-Shot	44.50	57.34	17.50	49.63
Dynamic Few-Shot (Best K)	50.00	63.40	32.50	82.00

Observations

- As prompting strategies progress from simple to structured, to static few-shot, and finally to dynamic few-shot, both **Execution Accuracy (Ex)** and **Valid Efficiency Score (VES)** steadily improve. This highlights the importance of providing richer contextual information to the model.

- For **Gemini**, dynamic few-shot prompting achieves the highest accuracy on both Spider and BIRD datasets, demonstrating its strong ability to adapt to relevant retrieved examples and utilize them effectively during SQL generation.

- **Llama** also benefits considerably from dynamic few-shot prompting, showing meaningful gains in both accuracy and validity compared to the baseline techniques, although the improvements are slightly smaller in magnitude than those of Gemini.

- When analysing different values of K during dynamic example retrieval, accuracy generally increases with larger K. On the **Spider dataset**, performance improves up to $K = 3$ and then plateaus, while VES continues rising up to $K = 5$. This indicates that additional examples help improve syntactic correctness even when execution accuracy saturates.

- On the **BIRD dataset**, improvements are more modest due to higher schema diversity. Both Ex and VES peak around $K = 3$ or $K = 4$, after which performance slightly declines, suggesting potential context saturation when too many examples are supplied.
- Overall, the best balance of accuracy and validity is achieved when K is in the range of 3 to 4. Too few examples limit reasoning ability, while too many may overload the model.

4 Discussion

4.1 Effectiveness of Dynamic Few-Shot

The dynamic retrieval strategy clearly demonstrated the advantages of in-context learning. Unlike the static few-shot baseline, which always relied on the same fixed set of examples, the dynamic method selected the most semantically similar examples to each incoming question. This allowed the model to see demonstrations that were much closer in structure to the actual query it needed to generate.

In practice, this led to noticeably more accurate SQL generation. The retrieved examples often matched the same reasoning pattern (join structure, aggregation, filtering conditions), so models could directly imitate the style of the nearest demonstrations. As a result, dynamic few-shot prompting consistently produced better queries than both the structured zero-shot and the static few-shot methods.

4.2 Dataset Impact

During evaluation, we observed meaningful differences between the Spider and BIRD datasets. Spider contains a broad range of academic-style schemas with clear foreign keys and relatively clean naming conventions. Many questions follow recognizable patterns (simple joins, grouping, filtering), which allowed both static and dynamic few-shot prompting to perform well. In general, the retrieved examples from Spider had higher semantic similarity with new questions, making dynamic retrieval more effective.

BIRD, on the other hand, has more realistic and noisy schemas. Relationships are sometimes implicit, column names are less standardized, and several queries involve multi-hop joins or complex aggregations. Because of this, the model occasionally struggled even when relevant examples were retrieved. The gap between static and dynamic few-shot prompting still existed, but both methods performed lower on BIRD than on Spider. This highlights that schema complexity and naming conventions directly affect the effectiveness of in-context learning.

4.3 Limitations

Although dynamic few-shot prompting significantly improved accuracy, it has a few limitations. First, the retrieval step itself introduces computational overhead since it requires embedding all example questions and computing similarity scores for every test input. For small-scale projects this is manageable, but for large-scale or real-time systems, this step could become expensive.

Second, the retrieval quality is heavily dependent on the embedding model. If the sentence embeddings fail to capture the true semantic similarity between questions, the system may retrieve inappropriate examples, which can mislead the LLM. In our setup, we used a fixed embedding model, which may introduce biases based on its training data.

Finally, because our experiments were restricted to approximately 200 examples due to API rate limits, the results should be interpreted as indicative rather than fully comprehensive. A larger-scale evaluation may further clarify the behavior of dynamic few-shot prompting across different database types.

5 Conclusion

This project explored different prompting strategies for the Text-to-SQL task, starting from simple zero-shot prompting and progressing through structured prompts, static few-shot prompting, and finally dynamic few-shot prompting. Across all experiments, dynamic retrieval emerged as the most effective approach. By selecting the most relevant examples for each question, the model received demonstrations that closely matched the desired SQL pattern, which significantly boosted execution accuracy.

Our findings show that the choice of the number of retrieved examples (K) is important. Too few examples limit the model's context, while too many can dilute the relevance of the prompt and lead to longer, less focused outputs. An intermediate value (in our project, around 3–5 examples) provided the best balance.

Overall, the Gemini model responded well to in-context learning and produced substantially improved SQL queries when helpful demonstrations were included. Although the evaluation set was limited by API constraints, the results strongly suggest that dynamic few-shot prompting is a practical and effective strategy for real-world Text-to-SQL systems.

References

[1] Bing Wang et al., *MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL*, arXiv:2312.11242, 2025. https://arxiv.org/pdf/2312.11242