SAFE-SQL: Self-Augmented In-Context Learning with Fine-grained Example Selection for Text-to-SQL

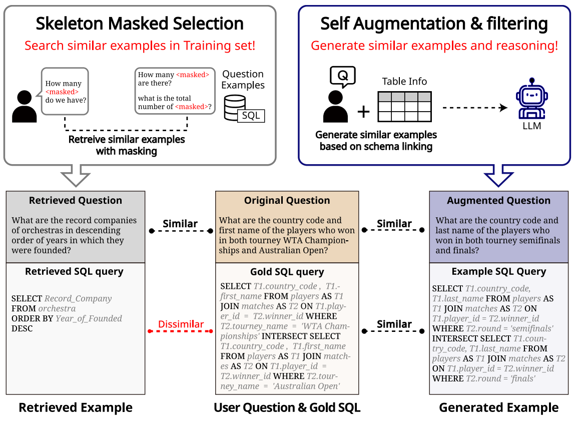
Anonymous ACL submission

Abstract

Text-to-SQL aims to convert natural language questions into executable SQL queries. While previous approaches, such as skeleton-masked selection, have demonstrated strong performance by retrieving similar training examples to guide large language models (LLMs), they struggle in real-world scenarios where such examples are unavailable. To overcome this limitation, we propose Self-Augmentation in-context learning with Fine-grained Example selection for Text-to-SQL (SAFE-SQL), a novel framework that improves SQL generation by generating and filtering self-augmented examples. SAFE-SQL first prompts an LLM to generate multiple Text-to-SQL examples relevant to the test input. Then SAFE-SQL filters these examples through three relevance assessments, constructing high-quality in-context learning examples. Using self-generated examples, SAFE-SQL surpasses the previous zero-shot, and few-shot Text-to-SQL frameworks, achieving higher execution accuracy. Notably, our approach provides additional performance gains in extra hard and unseen scenarios, where conventional methods often fail.

1 Introduction

Text-to-SQL generation converts questions into SQL queries that help users access information in databases. Traditional approaches on Text-to-SQL, such as, rule-based systems and early machine learning models, relied on hand-crafted rules or simple pattern matching to generate SQL queries. They often struggle with the ambiguity and context-dependence of natural language, making it challenging to accurately translate user intent into structured SQL commands (El Boujddaini et al., 2024; Mohammadjafari et al., 2025; Li and Jagadish, 2014). As the field progressed, more sophisticated approaches emerged, including skeleton-masked selection (Gao et al., 2023), relying on retrieving similar examples from training data to guide query


Skeleton Masked Selection
Search similar examples in Training set!
How many <masked> do we have?
How many <masked> are there?
what is the total number of <masked>?
Question Examples
SQL
Retreive similar examples with masking
Self Augmentation & filtering
Generate similar examples and reasoning!
Q
Table Info
LLM
Generate similar examples based on schema linking
Similar
Dissimilar
Similar
Similar
Retrieved Example
User Question & Gold SQL
Generated Example

Retrieved Question

What are the record companies of orchestras in descending order of years in which they were founded?

Retrieved SQL query

```
SELECT Record_Company
FROM orchestra
ORDER BY Year_of_Founded
DESC
```

Original Question

What are the country code and first name of the players who won in both tourney WTA Championships and Australian Open?

Gold SQL query

```
SELECT T1.country_code , T1.first_name FROM players AS T1 JOIN matches AS T2 ON T1.player_id = T2.winner_id WHERE T2.tourney_name = 'WTA Championships' INTERSECT SELECT T1.country_code , T1.first_name FROM players AS T1 JOIN matches AS T2 ON T1.player_id = T2.winner_id WHERE T2.tourney_name = 'Australian Open'
```

Augmented Question

What are the country code and last name of the players who won in both tourney semifinals and finals?

Example SQL Query

```
SELECT T1.country_code, T1.last_name FROM players AS T1 JOIN matches AS T2 ON T1.player_id = T2.winner_id WHERE T2.round = 'semifinals' INTERSECT SELECT T1.country_code, T1.last_name FROM players AS T1 JOIN matches AS T2 ON T1.player_id = T2.winner_id WHERE T2.round = 'finals'
```

Figure 1: The left example illustrates a failure in retrieving similar Text-to-SQL examples from the training set. In contrast, the right example demonstrates how our proposed self-augmented approach successfully generates similar examples autonomously.

generation. However, these methods face significant challenges in real-world scenarios where similar examples are often unavailable in the training set (Gan et al., 2021; Hong et al., 2024) or retrieve unrelated examples as shown in Figure 1. To overcome these problems, recent research has introduced to generate synthetic data. SQL-GEN, introduced by (Pourreza et al., 2024) introduces dialect-specific synthetic data to resolve the diverse SQL dialect challenges in Text-to-SQL systems. Another important aspect of synthetic data generation is incorporating key relationships from the schema and employing schema-distance-weighted column sampling (Zhao et al., 2022). However, these synthetic data generation methodologies predominantly require supervised fine-tuning, which demands substantial computational resources and time (Yang et al., 2024b). Moreover, self-generated examples can introduce significant noise and inaccuracies that undermine the quality of in-context learning. Errors in synthetic SQL queries or flawed reasoning paths may lead to incorrect interpretations of database schemas (Wretblad et al., 2024). As a result, relying on unfiltered self-generated examples for Text-to-SQL tasks can risk degrading overall model performance. Consequently, it is necessary to develop more efficient approaches that enhance the accuracy of Text-to-SQL while eliminating extra training costs and mitigating the adverse impacts of noisy self-generated examples.

In this paper, we propose SAFE-SQL, a novel approach that fully exploits the generative power of large language models (LLMs) to create high-quality synthetic examples in an unsupervised manner. SAFE-SQL enhances its inference capabilities without additional fine-tuning through four key steps: ***(1) Schema Linking***: Analyzing SQL test questions, database tables, and foreign keys to map relationships between queries and database structures ***(2) Example Generation***: Generating ten similar question-SQL query-reasoning path triplets per input using schema-linked information with LLMs ***(3) Threshold-based example selection***: Evaluating generated examples via embedding similarity, keyword & structural alignment, and reasoning path validity, retaining only those scoring above specific threshold and ***(4) Final SQL Inference***: Leveraging the curated examples, this step utilizes in-context learning to enhance the performance of large language models. This approach benefits from carefully selected examples that align with the natural language question and database schema, ensuring accurate and efficient SQL generation.

By relying on LLM-generated and filtered examples, SAFE-SQL significantly improves robustness and accuracy, particularly in complex or unseen scenarios where retrieval-based approaches struggle. Consequently, our approach eliminates the need for additional model training while achieving superior performance in Text-to-SQL tasks. Our contributions can be listed as follows:

- We propose SAFE-SQL, a fully unsupervised approach that leverages LLMs to generate synthetic examples without additional fine-tuning or reliance on predefined training sets.
- We introduce a structured filtering mechanism that selects high-quality question-SQL pairs based on embedding similarity, keyword and structural alignment, and reasoning path validation.
- Our method dynamically adapts examples using schema-linked information to boost SQL generation particularly in complex and unseen scenarios.

2 Related work

2.1 Supervised Fine-Tuning for Text-to-SQL

Supervised Fine-Tuning enables models to learn domain-specific nuances and schema alignments, significantly improving performance on specialized tasks (Sarker et al., 2024). Techniques such as schema linking (Yang et al., 2024c), constraint-based decoding (Scholak et al., 2021), and execution-guided generation (Wang et al., 2024) have further enhanced the robustness of fine-tuned models in handling domain-specific challenges. Synthesizes text-to-SQL data from weak and strong LLMs (Yang et al., 2024b) utilizes preference learning from the weak data from small LLMs and strong data from Large LLMs. However, supervised fine-tuning can be time-consuming and demands substantial computational resources. To remedy such issues, our approach sidesteps these heavy requirements by generating high-quality synthetic examples in a fully unsupervised manner.

2.2 In-Context Learning for Text-to-SQL

In-Context learning has emerged as another influential method, leveraging the ability of large language models to perform text to sql tasks by conditioning on a few examples provided in the input prompt, without requiring explicit parameter updates. Unlike traditional in-context learning methods that rely on a fixed set of examples, our approach dynamically generates and filters schema-aware examples to better adapt to varying query structures in real-world scenarios. AST-SQL (Shen et al., 2024) introduces using an abstract syntax tree algorithm to select similar examples and incorporate them into the in-context learning pipeline.

Skeleton Masked Similarity Method Skeleton masked similarity is an approach that emphasizes structural similarity between natural language questions and SQL queries. This method involves extracting the skeleton of a SQL query from the given question and masking unnecessary details to focus on its essential structure. By preserving key structural patterns, such as SELECT-FROM-WHERE clauses, this approach facilitates learning the correspondence between natural language expressions and SQL query elements. By moving beyond simple word-level matching, it captures deeper structural correlations, which is particularly effective for complex queries or linguistically diverse questions.

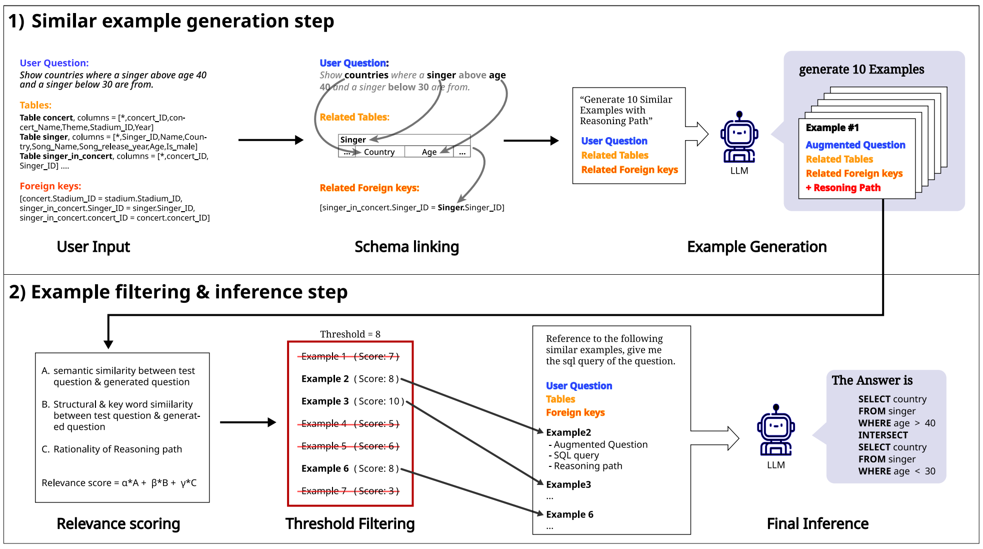

1) Similar example generation step
User Question:
Show countries where a singer above age 40 and a singer below 30 are from.
Tables:
Table concert, columns = [*,concert_ID,concert_Name,Theme,Stadium_ID,Year]
Table singer, columns = [*,Singer_ID,Name,Country,Song_Name,Song_release_year,Age,Is_male]
Table singer_in_concert, columns = [*,concert_ID, Singer_ID] ....
Foreign keys:
[concert.Stadium_ID = stadium.Stadium_ID, singer_in_concert.Singer_ID = singer.Singer_ID, singer_in_concert.concert_ID = concert.concert_ID]
User Input
User Question:
Show countries where a singer above age 40 and a singer below 30 are from.
Related Tables:
Singer
... Country Age ...
Related Foreign keys:
[singer_in_concert.Singer_ID = Singer.Singer_ID]
Schema linking
"Generate 10 Similar Examples with Reasoning Path"
User Question
Related Tables
Related Foreign keys
LLM
generate 10 Examples
Example #1
Augmented Question
Related Tables
Related Foreign keys
+ Resoning Path
Example Generation
2) Example filtering & inference step
A. semantic similarity between test question & generated question
B. Structural & key word simiilarity between test question & generated question
C. Rationality of Reasoning path
Relevance score = α*A + β*B + γ*C
Relevance scoring
Threshold = 8
~~Example 1 (Score: 7)~~
Example 2 (Score: 8)
Example 3 (Score: 10)
~~Example 4 (Score: 5)~~
~~Example 5 (Score: 6)~~
Example 6 (Score: 8)
~~Example 7 (Score: 3)~~
Threshold Filtering
Reference to the following similar examples, give me the sql query of the question.
User Question
Tables
Foreign keys
Example2
- Augmented Question
- SQL query
- Reasoning path
Example3
...
Example 6
...
LLM
The Answer is
SELECT country
FROM singer
WHERE age > 40
INTERSECT
SELECT country
FROM singer
WHERE age < 30
Final Inference

Figure 2: Overall flow of our proposed SAFE-SQL.

Classification and Decomposition Methods The classification and decomposition method simplifies SQL query generation by breaking down natural language questions into sequential steps. This process starts with classifying the input based on its structure or intent (e.g., single-table queries, multi-table joins, or nested queries) and then decomposes the question into smaller subtasks, such as identifying specific clauses and resolving their components. PTD-SQL (Luo et al., 2024) uses this step-wise approach to enhance LLM inference, reducing complexity and improving overall accuracy by modularizing the SQL generation process. While this method effectively handles complex and multi-intent queries, it struggles with questions that do not fit predefined categories.

Self-Training and Correction Methods Recent advancements in the Text-to-SQL domain have focused on enhancing model performance and reliability through self-training and self-correction techniques. These approaches enable models to generate data or correct errors autonomously, thereby improving data efficiency and generalization capabilities. LG AI research introduces Self-training strategy (Jo et al., 2024) utilizing pseudo-labeled unanswerable questions to develop a reliable Text-to-SQL system for Electronic Health Records (EHRs). SelECT-SQL (Shen and Kejriwal, 2024), proposes an innovative in-context learning solution that combines chain-of-thought prompting, self-correction, and ensemble methods. MAGIC (Askari et al., 2024), use multiple agents (manager, feedback, and correction agents) to iteratively refine generated SQL queries. Other approaches employ code language models to repeatedly correct errors in SQL queries (Chen et al., 2023). Unlike these iterative correction techniques, our approach preemptively filters and refines self-generated examples to reduce errors before query generation, thereby avoiding the need for additional correction steps.

3 Fine-grained Self-Augmentation for Text-to-SQL

We propose SAFE-SQL, a framework that automatically generates high-quality examples for in-context learning in Text-to-SQL tasks. Unlike traditional methods that rely on retrieving similar questions or using predefined templates, SAFE-SQL uses LLMs to create synthetic examples tailored to the given database schema. These examples are then filtered based on their semantic similarity, structural similarity, and the quality of their reasoning paths. Finally, we predict final SQL query for the test input using the self-generated examples via in-context learning.

3.1 Schema Linking

The first step in SAFE-SQL is schema linking, which identifies and extracts relevant schema elements from the database to reduce noise and improve performance in Text-to-SQL tasks (Cao et al., 2024). As shown in Figure 2, the schema linking step involves analyzing the test question to detect keywords and phrases that correspond to schema elements such as tables, columns, rows, and foreign keys within the database schema. This mapping narrows the focus to the most pertinent parts of the schema and provides the necessary context for generating examples that are both meaningful and grounded in the database structure.

3.2 Example Generation

Using the information obtained from schema linking, the LLM generates multiple synthetic examples for each test question. As illustrated in Figure 2, for each test question, we generate ten examples—each comprising a similar question, its corresponding SQL query, and a detailed reasoning path. These examples maintain structural similarity while varying elements such as numerical values, table names, and key attributes. This controlled variation ensures that the generated examples remain relevant while encouraging the model to generalize beyond surface-level patterns. By observing these modified instances, the model can infer the correct SQL query even when faced with unseen but structurally similar questions. In particular, the reasoning path outlines the logical steps required to derive the correct SQL query result, providing a comprehensive explanation of the query execution process. We provide the full prompt used for Large Language Models in Appendix B.1.

3.3 Relevance Scoring

After generating synthetic examples, SAFE-SQL evaluates examples, as shown in Figure 2. This evaluation process capture the core intent of the test question. To achieve this, each example e is assigned a composite relevance score Rel on a scale from 0 to 10, which is calculated as follows:

$$Rel = \alpha \cdot S(Q_e, Q_t) + \beta \cdot A(Q_e, Q_t) + \gamma \cdot R \quad (1)$$

Here, Q_t denotes the test question, Q_e denotes the generated example question, R denotes the reasoning path of the generated example, and the three components are defined as:

- $S(Q_e, Q_t)$**:** Semantic similarity between the generated question and test questions.
- $A(Q_e, Q_t)$**:** Structural alignment between the generated question and test questions.
- R**:** Evaluates the quality of the reasoning path accompanying the example. This score measures the completeness, logical consistency, and alignment of the reasoning steps with both the test question and the relevant schema elements.

The weighting factors α, β, and γ sum to 1, allowing SAFE-SQL to select examples. Only examples that surpass a predefined threshold are retained for guiding SQL query generation. We provide the full prompt used for LLMs in Appendix B.2. The semantic similarity score measures whether the generated question accurately preserves the intent of the test question, even if its wording differs. As shown in Figure 2, a generated question "Retrieve the list of countries where one singer is older than 40 and another is younger than 30." has a different structure but correctly maintains the meaning, resulting in a high semantic similarity score. Another generated question, "Show cities where a singer above age 40 and a singer below 30 are from." shares the same SQL structure as the test question, differing only in the geographical entity, resulting in a high structural similarity score. For reasoning path similarity, examples that follow a similar step-by-step logical process to derive the SQL query receive a higher score. Queries that require the same table joins, filtering conditions, and aggregation methods are more likely to align with the reasoning process of the test question, leading to a stronger match. These factors are complementary, and carefully evaluating these factors is crucial for effective example selection, ensuring that the retrieved examples are highly relevant to the test question.

3.4 Threshold Selection

To further ensure quality, SAFE-SQL retains only those examples with a relevance score above a predefined threshold θ. Formally, the set of selected examples is defined as:

$$E_{\text{selected}} = \{e \in E \mid Rel \geq \theta\} \quad (2)$$

where E represents all generated examples. This thresholding step filters out low-quality examples and ensures that only the most informative and contextually appropriate examples are used in the final inference. The threshold is set to 8, as Figure 4 demonstrates that this value provides an optimal balance between preserving high-quality examples and maintaining sufficient diversity for robust SQL generation.

3.5 Final Inference

In the final stage, the high-quality examples generated in previous steps are combined with the test question to construct a comprehensive prompt for the LLM. These examples, enriched with similar questions, corresponding SQL queries, and detailed reasoning paths, guide the LLM in generating the final SQL query. By integrating schema linking, synthetic example generation, relevance scoring, and threshold-based filtering, SAFE-SQL produces SQL queries that are both syntactically correct and semantically aligned with the intended database operations, while also providing an interpretable reasoning process.

4 Experiment

4.1 Experimental Setup

For our experiments, we employ six models for comparison purposes: GPT-4o (Hurst et al., 2024), GPT-4o-mini (Hurst et al., 2024), GPT-4 (Achiam et al., 2023), Llama-3.1-8B-Instruct (Dubey et al., 2024), Deepseek-coder-6.7b-instruct (Guo et al., 2024), Qwen2.5-7B-Instruct (Yang et al., 2024a), and starcoder2-7b (Lozhkov et al., 2024). The evaluation is conducted on the Spider (Yu et al., 2018) dev dataset, which is a widely used benchmark for Text-to-SQL systems. The Spider dev set is a large-scale, cross-domain benchmark specifically designed to assess the generalization capabilities of Text-to-SQL models. It contains 7,000 training samples covering 166 databases in various domains and 1,034 evaluation samples from 20 databases. Our analysis parts conduct with gpt-4o-2024-08-06 version model.

4.2 Baselines

We use the following baseline text to SQL methods: **Supervised fine tuning**, which fine-tune open source model, **Zero-shot**, which inference without examples, **Few-shot**, which inference with few examples. Synthesizes text-to-SQL data from weak and strong LLMs (Yang et al., 2024b) utilizes preference learning from the weak data from small LLMs and strong data from LLMs. SQL-Palm (Sun et al., 2024) introduces synthetic data augmentation to fine-tune open source models. Din SQL (Pourreza and Rafiei, 2023) breaking down the task into smaller sub-tasks, allowing large language models to iteratively improve their reasoning process through self-correction. C3 SQL (Dong et al., 2023) comprises Clear Prompting, Calibration with Hints, and Consistent Output, which systematically addresses model input, bias, and output to enhance performance using zero-shot prompt. Dail SQL (Gao et al., 2023) introduces effective few-shot learning which significantly reduces the number of tokens required per question. ACT-SQL (Zhang et al., 2023) enhances Text-to-SQL performance by automatically generating chain-of-thought exemplars, eliminating the need for manual labeling. PTD SQL (Luo et al., 2024) categorizing queries into subproblems and focusing on targeted drilling to improve large language models' reasoning capabilities. Unlike traditional few-shot methods that depend on human-selected examples, our Self-Augmented In-Context Learning method allowing the model to generate its own in-context examples, and used for final inference. This self-augmented approach removing the need for manually provided exemples while still leveraging the benefits of in-context learning, leading to improved adaptability and performance in Text-to-SQL tasks.

Method	**Model**	**Easy**	**Medium**	**Hard**	**Extra**	**All**
Supervised Fine-Tuning (SFT)						
SYN-SQL	Sense 13B	**95.2**	88.6	75.9	60.3	83.5
SQL-Palm	PaLM2	93.5	84.8	62.6	48.2	77.3
Zero-shot Methods						
Baseline	GPT-4	84.3	73.1	65.8	40.3	69.1
Baseline	GPT-4o	87.2	77.2	68.4	48.7	73.4
Baseline	GPT-4o-mini	84.8	75.6	67.0	46.1	71.5
C3q-SQL	GPT-4	90.2	82.8	77.3	64.3	80.6
Few-shot Methods						
DIN-SQL	GPT-4	91.1	79.8	64.9	43.4	74.2
DAIL-SQL	GPT-4	90.7	**89.7**	75.3	62.0	83.1
ACT-SQL	GPT-4	91.1	79.4	67.2	44.0	74.5
PTD-SQL	GPT-4	<u>94.8</u>	88.8	85.1	64.5	85.7
DEA-SQL	GPT-4	88.7	<u>89.5</u>	85.6	70.5	85.6
Self-augmented In-Context Learning						
SAFE-SQL	GPT-4	93.2	88.9	<u>85.8</u>	74.7	86.8
SAFE-SQL	GPT-4o	93.4	89.3	**88.4**	**75.8**	**87.9**
SAFE-SQL	GPT-4o-mini	93.6	87.5	86.1	<u>75.2</u>	<u>87.4</u>

Table 1: Execution accuracy across difficulty levels on the Spider development set. The highest score per row is in bold, and the second highest is underlined.

4.3 Evaluation Metrics

We use Execution Accuracy (EX) and Exact Match (EM) to evaluate the performance of our model. EX measures whether the SQL query generated by the model produces the same results as the ground truth query when executed on a database. Exact Match (EM), on the other hand, assesses whether the predicted SQL query exactly matches the ground truth query in its structure and syntax. By combining these two metrics, we ensure a comprehensive evaluation of both the correctness and execution reliability of the generated SQL queries.

Models	EX	EM
GPT-4o + SAFE	87.9	78.3
w/o Reasoning path	84.4 (-3.5)	73.6(-4.7)
w/o Relevance filtering	82.1 (-5.8)	68.5(-9.7)
w/o Schema linking	80.4 (-7.5)	65.1(-13.2)
w/o Similar examples	77.1 (-10.8)	61.9(-16.4)

Table 2: Ablation study results for SAFE-SQL, where removing each component leads to a performance drop.

4.4 Performance among SQL difficulty level

We analyze the performance of SAFE-SQL across different SQL difficulty levels and compare it with zero-shot, few-shot prompting methods, and supervised fine-tuning approaches. The results, presented in Table 1, demonstrate that SAFE-SQL achieves overall superior performance, with particularly strong improvements in hard and extra hard categories. Few-shot methods exhibit higher accuracy in Easy and Medium categories, which can be attributed to skeleton-masked selection which retrieves answers directly from the training set, leading to an inflated performance in simpler queries. SAFE-SQL excels in hard and extra hard categories, achieving significantly higher EX. This improvement is notably influenced by the inclusion of reasoning paths, which provide explicit guidance in SQL generation and enhance the model's ability to construct complex queries, as well as the filtering of misleading examples, which reduces potential confusion and prevents error propagation. While multiple factors contribute to SAFE-SQL's effectiveness, these mechanisms play a crucial role in enabling the model to generate more accurate and structurally sound SQL queries, especially in challenging scenarios where other approaches struggle.

4.5 Ablation study

To assess the contribution of each key component in our model, we conduct an ablation study by systematically removing four critical modules: **Reasoning Path**, **Relevance Score**, **Schema Linking**, and **Similar Examples**. We evaluate the resulting impact on performance using EX shown in Table 2. Our findings indicate that each component plays a crucial role in the model's effectiveness. Removing the Reasoning Path leads to a 3.5-point drop in EX, highlighting its importance in guiding the model toward generating accurate SQL queries. The absence of the Relevance Score resulted in a 5.8-point decrease in EX, underscoring its contribution to overall performance. Eliminating Schema Linking causes a 7.5-point drop in EX, which demonstrates its critical role in similar example construction. Overall, each of the four components—Reasoning Path, Relevance Score, Schema Linking, and Similar Examples—is essential for achieving optimal performance in SQL generation.

4.6 Analysis

Score	cos θ	# of Generated EX	% Filtered EX
≥ 0	0.581	10340	0 %
≥ 2	0.625	10185	1.50% (-155)
≥ 4	0.744	9883	4.41% (-457)
≥ 6	0.762	9378	9.30% (-962)
≥ 8	0.765	8606	16.76% (-1734)
≥ 10	0.769	6795	34.28% (-3545)

Table 3: Summary of data generation, filtering results, and embedding similarity analysis by score.

Number of generated and filtered examples per score, along with an embedding similarity analysis of the filtered examples For each test question in the Spider dev set, 10 examples are generated, resulting in a total of 10,340 examples. The quality of these examples is assessed using a relevance score ranging from 0 to 10. As shown in Table 3, the 65.71% of examples are assigned a score of 10, while the 0.59% of examples are received a score of 0. This trend suggests that the LLM tends to assign high relevance to its own generated examples. The similarity is computed using cosine similarity, where higher scores indicate greater semantic alignment between the test questions and the retained examples. As the filtering threshold increases, the embedding similarity also increases, suggesting that higher-relevance examples exhibit stronger semantic consistency with the test questions. However, we also observe that overly strict filtering—selecting only examples with a perfect score of 10—leads to a decline in performance. This drop occurs because an excessively high threshold significantly reduces the number of available examples, limiting the diversity.

Effect of question embedding similarity on Execution Accuracy. In Figure 3, the left graph illustrates the correlation between embedding similarity and EX. Each point represents one of the 11 data points obtained by filtering examples based on different threshold scores (0 to 10). The data points follow an upward trend, suggesting that higher similarity tends to result in better EX. The red line indicates the overall correlation, with a coefficient

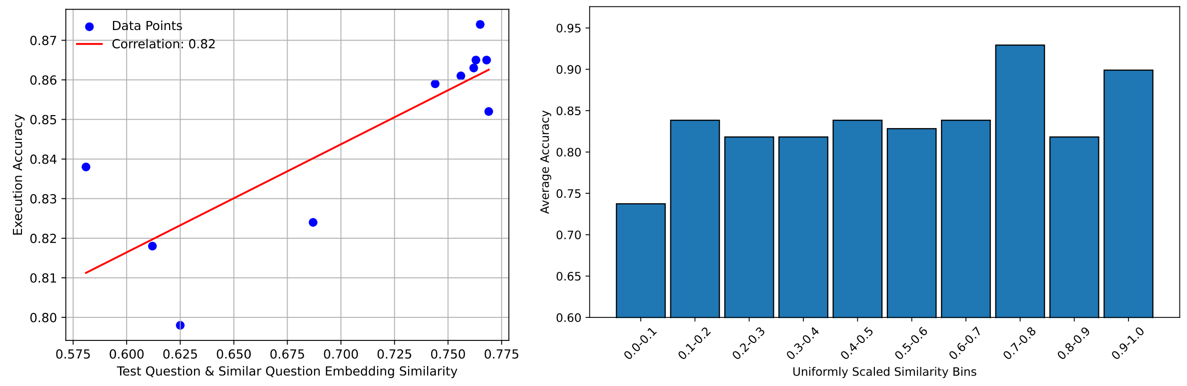


Figure 3: (Left) Correlation between question embedding similarity and average EX, (Right) Average EX across embedding similarity bins

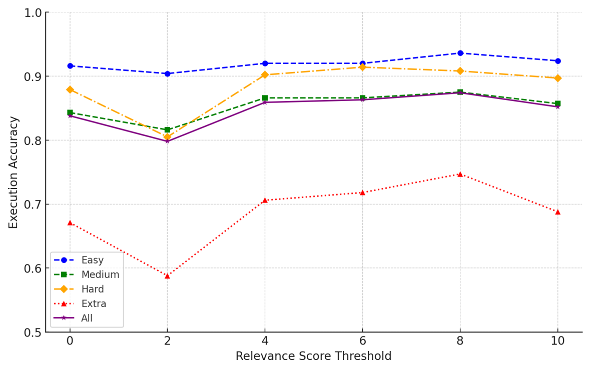


Figure 4: Performance of GPT-4o at different relevance score thresholds.

of 0.82, showing a relatively strong positive relationship. Building on this analysis, the right graph provides a more fine-grained view by examining the execution accuracy of individual generated examples based on their embedding similarity with test questions. The x-axis represents the normalized similarity between the test question and the generated question, and the y-axis indicates EX. The results show that EX is lowest in the 0.0-0.1 similarity range, suggesting that examples with very low similarity to test questions tend to be less useful. As similarity increases, EX generally improves, peaking in the 0.7-0.8 range. This suggests that examples with a moderate to high similarity to test questions are more effective in generating executable SQL queries. However, accuracy drops slightly in the 0.8-0.9 range before rising again in the 0.9-1.0 range. This indicates that excessively high similarity can reduce diversity, potentially limiting the model's generalization ability.

Effect of Relevance Scoring Thresholds on Performance. To further evaluate the effectiveness of SAFE-SQL, we conduct a detailed case study using varying thresholds for the relevance scoring mechanism as shown in Figure 4. The self-generated examples are filtered based on relevance scores, with thresholds ranging from 0 to 10. For each test question, the number of high-scoring examples varied due to the specific content and schema structure (e.g., some test questions had six examples with scores ≥ 8, while others had three). The selected examples are then used during the final inference stage to generate SQL queries. The ≥ 8 threshold consistently produced the best results, validating the robustness of SAFE-SQL's relevance score filtering. The results demonstrate that selecting high-quality examples plays a critical role in guiding LLMs to generate accurate SQL queries, regardless of the underlying model.

Effect of three measuring components on Performance. To assess the impact of the three measuring components—semantic similarity (α), keyword & structural similarity (β), and reasoning path quality (γ)—on EX, we conduct experiments by varying their respective weightings. The results, presented in Table 5, highlight distinct performance trends across different difficulty levels. Notably, the exclusion of reasoning path quality leads to a drop in EX, particularly in the Hard and Extra Hard. This suggests that a well-structured reasoning path is crucial for handling complex queries, as it provides essential logical steps that bridge the gap between natural language understanding and SQL formulation. Conversely, semantic similarity and structural SQL query similarity have a greater influence on the Easy and Medium levels. This is because these queries tend to be relatively straightforward, meaning that having structurally

SQL Question	GOLD SQL Query	Augmented SQL Question	Generated Reasoning Path	Relevance Score
Question1: What are the names, countries, and ages for every singer in descending order of age?	`SELECT name, country, age FROM singer ORDER BY age DESC`	What are the names, ages, and countries of all singers from a specific country, sorted by age in descending order?	1. Identify the desired columns: name, age, and country. 2. Specify the table: singer. 3. Sort the results by age in descending order.	Semantic similarity = 10 Structural Similarity = 10 Reasoning path = 10 Relevance score = (10+10+10)/3 = 10
Question2: Return the names and template ids for documents that contain the letter w in their description.	`SELECT document_name, template_id FROM Documents WHERE Document_Description LIKE "%w%"`	Retrieve the titles and category IDs of articles whose summaries contain the word "data".	1. Identify the necessary columns: extract title and category_id from the Articles table. 2. Locate the relevant table: use the Articles table as it contains the required data. 3. Define the filtering condition: apply a WHERE clause to check if the summary column contains the substring "data". 4. Use the LIKE operator: employ LIKE '%data%' to search for any instance of "data" within the summary. 5. Retrieve the results: return the title and category_id values for all matching records.	Semantic similarity = 7 Structural Similarity = 9 Reasoning path = 8 Relevance score = (7+9+8)/3 = 8
Question3: What is the number of car models that are produced by each maker and what is the id and full name of each maker?	`SELECT Count(*), T2.FullName, T2.id FROM MODEL_LIST AS T1 JOIN CAR_MAKERS AS T2 ON T1.Maker = T2.id GROUP BY T2.id;`	List all employees who work in the IT department along with their employee ID and hire date.	1. Identify required details: employee ID and hire date. 2. Filter condition: find employees who work in IT. 3. Retrieve data: select only emp_id and hire_date.	Semantic similarity = 6 Structural Similarity = 3 Reasoning path = 2 Relevance score = (6+3+2)/3 = 3.67

Table 4: Examples of original and augmented SQL questions with reasoning paths by GPT-4o.

α	β	γ	Easy	Medium	Hard	Extra	EX
0.33	0.33	0.33	**93.4**	**89.3**	88.4	**75.8**	**87.9**
1	0	0	90.7	84.2	82.3	68.3	82.8
0	1	0	89.8	85.6	81.2	69.2	83.1
0	0	1	89.2	85.1	84.3	71.7	83.7
0.5	0.5	0	91.2	87.3	82.5	69.4	84.4
0.5	0	0.5	92.5	87.9	83.5	70.3	85.3
0	0.5	0.5	92.7	86.8	**88.5**	72.4	86.1

Table 5: Execution accuracy across difficulty levels under different weights: semantic similarity (α), Structural similarity (β), and reasoning path quality (γ).

similar SQL questions in the example set often provides sufficient guidance for generating correct queries. In these cases, direct pattern matching and schema alignment play a larger role. Overall, the findings demonstrate that a balanced combination of all three components is essential for optimizing performance across different levels of query complexity.

4.7 Case Study

As shown in Table 4, test questions from the Spider dev set alongside their generated similar examples, evaluated based on semantic similarity, structural similarity, and the reasoning path score, which together determine the relevance score. The first example achieves a perfect relevance score of 10, as the generated question closely aligns with the original in meaning, structure, and reasoning. The SQL formulation remains nearly identical, and the reasoning path explicitly details each step, ensuring full alignment. The second example receives a relevance score of 8, with semantic similarity of 7 due to minor differences in terminology ("documents" vs. "articles" and "letter 'w'" vs. "word 'data'"). However, its structural similarity remains high, as the SQL structure is nearly identical. The reasoning path score of 8 reflects a clear explanation of query formulation, though slightly less detailed than the first example. The third example has the lowest relevance score due to significant differences. The generated question shifts focus from counting car models to listing IT employees, resulting in semantic similarity of 6 and structural similarity of 3. These results emphasize the importance of fine-grained example selection due to the varing quality of generated examples.

5 Conclusion

In this paper, we introduce SAFE-SQL, a novel unsupervised framework designed to improve Text-to-SQL execution accuracy by generating and filtering self-augmented examples. Through extensive experiments, we show that fine-grained example generation, and optimal threshold filtering contribute to the overall performance increase. Specifically, our method achieves state-of-the-art results, with notable improvements over ablated versions, highlighting the importance of these modules in generating accurate and semantically valid SQL queries. Our findings underscore the capability of large language models when carefully structured and enhanced, to address complex Text-to-SQL tasks.

Limitations

While SAFE-SQL demonstrates strong performance in generating accurate and semantically valid SQL queries, there are a few limitations that should be addressed in future work. Although the model performs well on the tested datasets, its ability to generalize to highly diverse or domain-specific SQL tasks remains to be fully evaluated. The current framework also relies on large language models like GPT-4o, which may not be easily scalable to low-resource settings or environments with limited computational resources. Handling edge cases and extremely complex queries, which might require deeper schema understanding and more sophisticated reasoning, is another challenge for the model.

Ethics Statement

This research introduces SAFE-SQL, a self-augmented in-context learning framework for Text-to-SQL tasks. While our approach enhances SQL generation without additional fine-tuning, it relies on LLMs, which may inherit biases from training data. We mitigate potential biases and inaccuracies through structured filtering and relevance scoring. Our study uses publicly available datasets, ensuring compliance with data privacy standards. We encourage responsible use of our method, particularly in applications requiring high accuracy and fairness.

References

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

Arian Askari, Christian Poelitz, and Xinye Tang. 2024. Magic: Generating self-correction guideline for in-context text-to-sql. *Preprint*, arXiv:2406.12692.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. Rsl-sql: Robust schema linking in text-to-sql generation. *Preprint*, arXiv:2411.00073.

Ziru Chen, Shijie Chen, Michael White, Raymond Mooney, Ali Payani, Jayanth Srinivasa, Yu Su, and Huan Sun. 2023. Text-to-sql error correction with language models of code. *Preprint*, arXiv:2305.13073.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Farida El Boujddaini, Ahmed Laguidi, and Youssef Mejdoub. 2024. A survey on text-to-sql parsing: From rule-based foundations to large language models. In *International Conference on Connected Objects and Artificial Intelligence*, pages 266–272. Springer.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021. Exploring underexplored limitations of cross-domain text-to-sql generalization. *Preprint*, arXiv:2109.05157.

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

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

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-generation database interfaces: A survey of llm-based text-to-sql. *Preprint*, arXiv:2406.08426.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Yongrae Jo, Seongyun Lee, Minju Seo, Sung Ju Hwang, and Moontae Lee. 2024. Lg ai research kaist at ehrsql 2024: Self-training large language models with pseudo-labeled unanswerable questions for a reliable text-to-sql system on ehrs. *Preprint*, arXiv:2405.11162.

Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Ruilin Luo, Liyuan Wang, Binghuai Lin, Zicheng Lin, and Yujiu Yang. 2024. Ptd-sql: Partitioning and targeted drilling with llms in text-to-sql. *Preprint*, arXiv:2409.14082.

Ali Mohammadjafari, Anthony S. Maida, and Raju Gottumukkala. 2025. From natural language to sql: Review of llm-based text-to-sql systems. *Preprint*, arXiv:2410.01066.

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

Mohammadreza Pourreza, Ruoxi Sun, Hailong Li, Lesly Miculicich, Tomas Pfister, and Sercan O. Arik. 2024. Sql-gen: Bridging the dialect gap for text-to-sql via synthetic data and model merging. *Preprint*, arXiv:2408.12733.

Shouvon Sarker, Xishuang Dong, Xiangfang Li, and Lijun Qian. 2024. Enhancing llm fine-tuning for text-to-sqls by sql quality measurement. *arXiv preprint arXiv:2410.01869*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *Preprint*, arXiv:2109.05093.

Ke Shen and Mayank Kejriwal. 2024. Select-sql: Self-correcting ensemble chain-of-thought for text-to-sql. *Preprint*, arXiv:2409.10007.

Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, Kaustubh Vyas, Yuanyi Ji, and Jeff Z. Pan. 2024. Improving retrieval-augmented text-to-sql with ast-based ranking and schema pruning. *Preprint*, arXiv:2407.03227.

Ruoxi Sun, Sercan Ö. Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *Preprint*, arXiv:2306.00739.

Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. 2024. Tool-assisted agent on sql inspection and refinement in real-world scenarios. *arXiv preprint arXiv:2408.16991*.

Niklas Wretblad, Fredrik Gordh Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. 2024. Understanding the effects of noise in text-to-sql: An examination of the bird-bench benchmark. *Preprint*, arXiv:2402.12243.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024b. Synthesizing text-to-sql data from weak and strong llms. *Preprint*, arXiv:2408.03256.

Sun Yang, Qiong Su, Zhishuai Li, Ziyue Li, Hangyu Mao, Chenxi Liu, and Rui Zhao. 2024c. Sql-to-schema enhances schema linking in text-to-sql. In *International Conference on Database and Expert Systems Applications*, pages 139–145. Springer.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought. *Preprint*, arXiv:2310.17342.

Yiyun Zhao, Jiarong Jiang, Yiqun Hu, Wuwei Lan, Henry Zhu, Anuj Chauhan, Alexander Li, Lin Pan, Jun Wang, Chung-Wei Hang, Sheng Zhang, Marvin Dong, Joe Lilien, Patrick Ng, Zhiguo Wang, Vittorio Castelli, and Bing Xiang. 2022. Importance of synthesizing high-quality data for text-to-sql parsing. *Preprint*, arXiv:2212.08785.

A Appendix

B Prompts for SAFE-SQL

B.1 Prompt for example generation.

For example generation, we use zero shot prompt as shown in the figure 6.

You are a powerful text-to-SQL reasoner. Your task is to generate ten similar questions, ten SQL queries, and ten reasoning paths for how the SQL queries are derived. To ensure high-quality examples, focus on the following three key aspects:

Semantic Similarity
Ensure that all generated questions have the same underlying meaning as the test question. Variations in wording, synonyms, and phrasing are allowed as long as they preserve the intended query objective. Avoid introducing ambiguity or additional constraints that alter the intent.
Structural Similarity
While key terms (such as table names, column names, and numerical values) may vary, their functional roles and relationships should remain intact.
Reasoning Path Similarity
The logical reasoning required to construct the SQL query should remain consistent across examples.Clearly outline each step, including how key conditions are identified and mapped to SQL operations.Maintain coherence in how joins, aggregations, filters, and sorting operations are applied. Do not explain me about the result and just give me ten examples.

## Schema linking: schema_linking[i]
## Tables: test_table[i]
## Foreign keys: test_foreign_keys[i]
## Question: test_question[i]

## Similar Question:
## SQL query:
## Reasoning Path:

Table 6: The zero-shot prompt used for example generation

B.2 Prompt for filtering examples.

For example generation, we use zero shot prompt as shown in figure 7.

B.3 Prompt for final inference.

For final inference, we use zero shot prompt as shown in figure 8.

C Impact of model size

Performance based on generated examples across different model size As shown in Ta-

You are a powerful text-to-SQL reasoner. Given a test question and a set of examples, compute the relevance score for each example based on the following criteria. Do not explain me about the answer, just give me scores.

Semantic Similarity of Questions
Compare the overall meaning of the test question and the example question. Higher scores should be assigned if the two questions have the same intent, even if they are phrased differently. Consider synonyms, paraphrasing, and minor wording variations that do not alter the fundamental meaning. Assign lower scores if the test and example questions focus on different database operations (e.g., aggregation vs. filtering) or require fundamentally different types of information.(up to 10 points).
10: Almost identical meaning and intent.
7–9: Minor paraphrasing but highly relevant.
4–6: Some overlap but different focus.
1–3: Mostly unrelated meaning.
0: Completely different intent.

Keyword & Structural Similarity
Evaluate the structural alignment between the test question and the example question by analyzing how key elements (such as entities, attributes, and numerical values) are connected. Even if individual nouns, verbs, or numbers differ, the overall relational structure should be considered. Focus on whether the dependencies between key components (e.g., how entities relate to each other in the database) remain consistent.(up to 10 points).
10: Nearly identical structural relationships and dependencies.
7–9: Mostly similar structure, with minor differences in entity connections.
4–6: Some overlap, but noticeable differences in how key components interact.
1–3: Few shared structural relationships, making alignment weak.
0: No meaningful structural similarities.

Reasoning Path Similarity
Evaluate whether the logical steps needed to answer the example question align with those required for the test question. Consider whether the database operations (e.g., filtering, aggregation, joins, subqueries) are similar.A high score should be given if the example follows the same logical sequence to derive the SQL query.Lower scores should be assigned if the reasoning process differs significantly, even if the questions seem similar at a surface level.(up to 10 points).
10: Exact reasoning process to get right SQL query.
7–9: Mostly similar but with minor differences.
4–6: Some alignment but different key steps.
1–3: Largely different reasoning.
0: Completely unrelated logic.

## Question: test_question[i]
## Similar Question: similar_question[i]
## Reasoning Path: reasoning_path[i]
## Relevance score:

Table 7: The zero-shot prompt used for filtering examples.

You are a powerful text-to-SQL reasoner. Your task is to generate the final SQL query using a set of selected examples that provide guidance on query construction. Utilizing Selected Examples. Do not explain me about the answer, just give me SQL query.
A set of chosen examples, each containing: A natural language question similar to the test question A corresponding SQL query A detailed reasoning path explaining how the SQL query was derived These examples are selected based on three key criteria:

Semantic Similarity of Questions The selected examples closely match the intent of the test question. Variations in wording do not change the meaning.
Structural Similarity The database schema elements (tables, columns, joins) used in the examples align with the test question. The SQL syntax and structure are relevant to the expected query.
Reasoning Path Similarity The logical steps used to construct the SQL query align with the reasoning required for the test question. Key transformations, filtering conditions, and aggregation logic are similar.
Final SQL Query Construction
Using the selected examples, generate the final SQL query that correctly retrieves the desired result for the given test question. Follow the reasoning patterns observed in the examples. Maintain correct table joins, filters, aggregations, and conditions based on schema constraints. Ensure that the final query accurately represents the intent of the test question while leveraging the insights from the selected examples. Now, generate the final SQL query for the given test question:

##Tables: test_table[i]
##Foreign_keys: test_foreign_keys[i]
##Question: text_question[i]
##Filtered_example: filtered_example[i]

Table 8: The zero-shot prompt used for Final SQL query inference.

	Easy	Med	Hard	Extra	All
Qwen 2.5-3B	62.4	61.2	58.6	48.8	59.1
Qwen 2.5-7B	80.0	78.0	67.2	51.8	72.3
Qwen 2.5-14B	**81.2**	**80.3**	**69.5**	**56.4**	**74.7**

Table 9: Execution accuracy performance of different size of models of Qwen series across difficulty levels of spider dev set.

ble 9, We investigate the impact of model size on example generation with different variants of the Qwen2.5 Models. The results demonstrate that the 14B model achieves the highest overall performance, followed by the 7B and the 3B. This trend is consistent across all difficulty levels, with large model size generating higher-quality examples that lead to more accurate SQL query generation. The performance improvement with increasing model size can be attributed to the enhanced capacity of larger models to capture SQL question patterns and semantic relationships. Moreover, larger models possess more extensive information, allowing them to generate more appropriate questions and construct detailed reasoning paths, which contribute to the overall accuracy of SQL query generation.

D Spider dev training set embedding clusters.

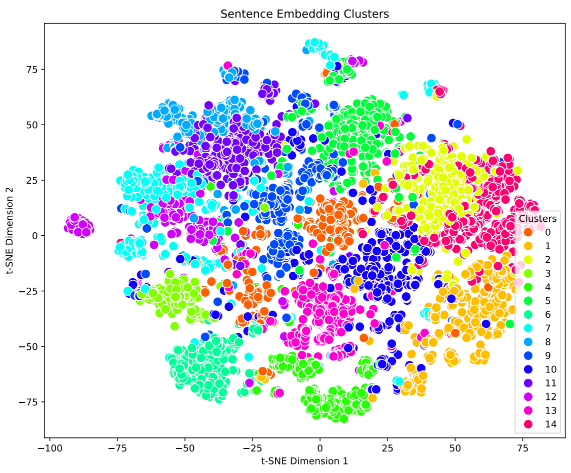


Figure 5: Embedding of spider dev set training questions.

Although questions within the same category share semantic similarities, they may belong to different clusters, leading to inconsistencies when retrieving examples from the training set. This highlights the limitations of training set retrieval in Text-to-SQL tasks.

Methods	**Model**	**Type**	**Easy**	**Medium**	**Hard**	**Extra**	**All**
SAFE-SQL	Llama3.1-8B-Instruct	ICL	73.2%	76.1%	63.2%	59.4%	70.5%
SAFE-SQL	Deepseek-coder-6.7B	ICL	88.8%	65.5%	63.8%	25.3%	64.2%
SAFE-SQL	Qwen2.5-7B-Instruct	ICL	83.6%	80.7%	78.7%	69.4%	79.2%
SAFE-SQL	Starcoder-7B	ICL	89.2%	88.9%	84.5%	70.6%	85.2%

Table 10: Execution accuracy performance of different methods across difficulty levels of spider dev set.

D.1 Additional model performance

To evaluate the impact of example generation quality on Text-to-SQL performance, we conducted experiments using different models for final inference. Examples generated by GPT-4o, followed by inference using the target model. Large language models', such as Qwen 2.5-7B and Deepseek-coder-6.7B, ability to generate high-quality, semantically relevant in-context examples is limited. To mitigate this, we first used GPT-4o to generate in-context examples and filtering examples, then performed final inference using the selected model. Our results show that leveraging GPT-4o for example generation and scoring improved overall execution accuracy by 6.9 points compared to fully relying on Qwen 2.5-7B for the entire process as shown in Table 9. This confirms that high-quality, well-aligned in-context examples play a crucial role in enhancing Text-to-SQL performance, especially in complex queries.