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

Jimin Lee[1], **Ingeol Baek**[1], **Byeongjeong Kim**[1], **Hyunkyung Bae**[2], **Hwanhee Lee**[1]*
[1]Department of Artificial Intelligence, Chung-Ang University, [2]New York University
{ljm1690, ingeolbaek, michael97k, hwanheelee}@cau.ac.kr, hb3135@nyu.edu

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 **S**elf-**A**ugmentation in-context learning with **F**ine-grained **E**xample selection for Text-to-**SQL** (SAFE-SQL), a novel unsupervised framework that enhances SQL generation by generating and intelligently filtering self-augmented examples. SAFE-SQL leverages an LLM to generate diverse Text-to-SQL examples, which are then filtered by a novel fine-grained mechanism using criteria for semantic similarity, structural alignment, and reasoning path quality to curate high-quality in-context learning examples. Leveraging these carefully selected self-generated examples, SAFE-SQL significantly surpasses previous zero-shot and few-shot Text-to-SQL frameworks, achieving superior execution accuracy. Notably, our approach demonstrates substantial performance gains in challenging extra hard and unseen scenarios, where conventional methods often struggle.

1 Introduction

Text-to-SQL generation converts questions into SQL queries that help users access information in databases. Traditional approaches on Text-to-SQL rely 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).

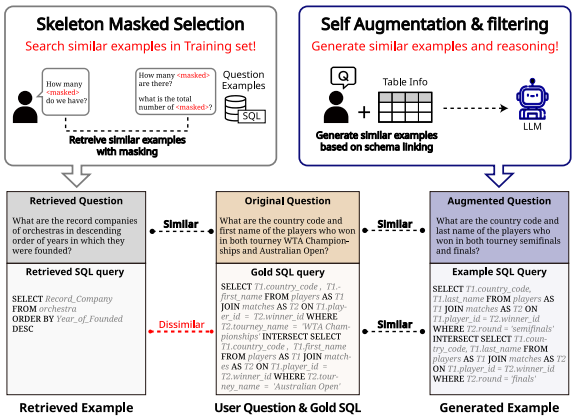

Figure 1: The example on the left shows a failure in retrieving relevant examples due to masked keywords, which results in superficially similar but actually unrelated questions being selected. In contrast, our self-augmented approach generates N-examples and filters them using 3 criteria, resulting in appropriate example retrieval.

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 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 unrelated examples are retrieved as shown in Figure 1. To overcome these problems, recent research has introduced methods to generate synthetic data. SQL-GEN, presents 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.,

*Corresponding author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 19035–19047
November 4-9, 2025

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 pose a risk of 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 by implementing a robust filtering mechanism.

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 N-question-SQL query-reasoning path triplets per input using schema-linked information with LLMs ***(3) Threshold-based example selection***: Filtering generated examples using specifically designed relevance criteria based on semantic similarity, Structural alignment, and reasoning path validity, retaining only those scoring above a specific threshold to ensure high quality and relevance for in-context learning examples 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. 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.
- Our method leverages schema linking to dynamically adapt examples, boosting the performance of Text-to-SQL in complex scenarios.
- We introduce a structured filtering mechanism that selects high-quality question-SQL pairs based on semantic similarity, structural alignment, and reasoning path validation.

2 Related Work

Structural and Semantic Information for Text-to-SQL Advances in Text-to-SQL have increasingly emphasized the importance of effectively utilizing structural and semantic information derived from the database schema. RAT-SQL (Wang et al., 2021) introduces relation-aware transformer architectures capable of encoding both the natural language question and the complex structure of the database schema, leading to improved schema linking performance. Concurrently, PICARD (Scholak et al., 2021) demonstrates that leveraging constrained decoding with step-by-step execution during generation can reduce the likelihood of producing invalid SQL queries. Rather than treating SQL generation as a pure sequence prediction task, PICARD executes partial SQL statements during generation, thus enforcing syntactic and semantic correctness. Both of these approaches highlight the importance of integrating structural and semantic information to generate correct SQL queries. Building on this line of research, our work explores how self-augmented examples can effectively incorporate structurally and semantically relevant information for in-context learning, particularly in scenarios where large annotated datasets are unavailable.

In-context Learning with Example Augmentation and Filtering As LLMs have demonstrated strong performance in in-context learning settings, recent work has focused on improving the effectiveness of demonstrations through better example augmentation and selection (Toteja et al., 2025). Self-Instruct (Wang et al., 2023) introduces a framework for generating instruction-tuning data by prompting the model to synthesize and filter examples, leveraging synthetic supervision. Further studies on demonstration selection for in-context learning (Wang et al., 2024) have systematically studied strategies for selecting effective in-context examples, including similarity-based retrieval and clustering methods. Integrating self-generated prompts with explicit reasoning

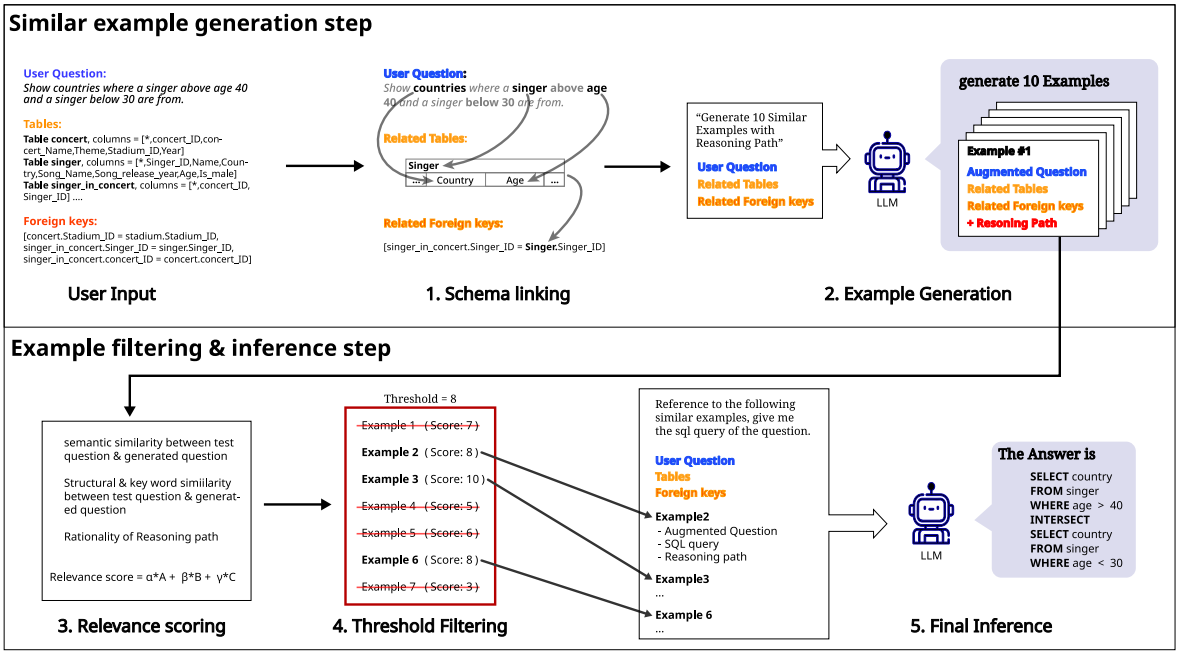

Figure 2: Overall flow of our proposed SAFE-SQL.

chains has also been shown to significantly improve in-context learning outcomes by guiding the model's thought process (Shum et al., 2023; Wei et al., 2023). These studies collectively underscore the impact of demonstration quality on LLM performance and highlight the potential of intelligently curating examples. Focusing on the Text-to-SQL task, our work distinguishes itself by generating synthetic examples and filtering them using a novel fine-grained mechanism that considers semantic similarity, structural similarity, and reasoning path quality. This allows us to generate multiple candidate examples and select the most effective through this tailored fine-grained filtering process.

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 alignment*, and the *quality of reasoning paths*. Finally, we predict the 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 relevant 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 a pool of 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. The generated SQL questions maintain structural similarity while varying elements such as numerical values, table names, and key attributes. This 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 face 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 LLMs in Appendix B.1.

3.3 Relevance Scoring

After generating a set of synthetic examples, SAFE-SQL employs a crucial evaluation process rooted in novel fine-grained example selection to determine the relevance of each generated example to the test question. This fine-grained selection process is integral to our method, ensuring that only high-quality, contextually appropriate examples are used for in-context learning, moving beyond simple retrieval to curate examples truly relevant and beneficial for the Text-to-SQL task.

To achieve this, we assign composite relevance score Rel on a scale from 0 to 10 to each example e, 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 represents the test question, Q_e denotes the generated example question. The coefficients α, β, and γ are weighting factors that sum to 1, allowing for adjustment of the relative importance of each component in the fine-grained selection score. The three components are defined as follows:

- **Semantic Similarity** $S(Q_e, Q_t)$**:** assesses if the generated question preserves the underlying meaning and intent to ensure the example aligns with the user's core query objective.
- **Structural Alignment** $A(Q_e, Q_t)$**:** evaluates structural correspondence based on key database elements and their relationships, which is important for mapping natural language to a similar database structure and operations.
- **Reasoning Path Quality** R**:** evaluates the alignment of the example's logical derivation steps and database operations (e.g., filtering, aggregation, joins, subqueries) with the test question's required logic.

We utilize LLMs to compute the score for each of the three components based on our specifically designed instructions and criteria. Specifically, the LLM applies a predefined, multi-point scoring rubric (detailed in Appendix B.2) to assign a quantitative score (0-10) for each criterion. This process allows for a nuanced assessment of the degree of alignment between the generated example and the test question along each dimension, moving beyond simple binary or qualitative judgments. By carefully evaluating these three factors through our fine-grained example selection process, SAFE-SQL ensures that the selected examples are highly relevant and informative, contributing to more accurate and effective SQL query generation.

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 filtered 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-3.1-70B-Instruct (Dubey et al., 2024), Llama3.3-70B-Instruct (Dubey et al., 2024),

Qwen2.5-72B (Yang et al., 2024a), Gemma3-12b (Team et al., 2025), and Gemma3-27b (Team et al., 2025). The evaluation is conducted on the Spider dev dataset (Yu et al., 2018) and the Bird dev dataset (Li et al., 2023), which are widely used benchmarks for Text-to-SQL systems. The Spider dev set contains 7,000 training samples covering 166 databases in various domains and 1,034 evaluation samples from 20 databases, comprised of four difficulty levels. BIRD is a large cross-domain Text-to-SQL dataset with 12,751 question-SQL pairs across 95 databases. Since the test sets of both the Spider and BIRD datasets are only accessible through specific evaluation servers, we conduct our evaluation using their respective development sets.

4.2 Baselines

We use the following baseline Text-to-SQL methods: **Supervised fine tuning**, which fine-tunes an open source model, **Zero-shot**, which infers without examples, **Few-shot**, which infers with few examples. **Synthesizing 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 improve their reasoning process through self-correction iteratively. **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 the zero-shot prompt. **Dail-SQL** (Gao et al., 2023) introduces effective few-shot learning, significantly reducing 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) categorizes queries into subproblems and focuses on targeted drilling to improve LLMs' reasoning capabilities.

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.

Method	Model	Easy	Medium	Hard	Extra	All	Time
Supervised Fine-Tuning (SFT)							
SYN-SQL	Sense-13B	**95.2**	88.6	75.9	60.3	83.5	-
SQL-PaLM	PaLM-2	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	1.28
Baseline	GPT-4o	87.2	77.2	68.4	48.7	73.4	0.93
Baseline	GPT-4o-mini	84.8	75.6	67.0	46.1	71.5	1.07
C3-SQL	GPT-4	90.2	82.8	77.3	64.3	80.6	19.34
Few-shot Methods							
DIN-SQL	GPT-4	91.1	79.8	64.9	43.4	74.2	4.37
DAIL-SQL	GPT-4	91.9	**90.1**	75.2	63.8	83.6	16.79
ACT-SQL	GPT-4	91.1	79.4	67.2	44.0	74.5	4.55
PTD-SQL	GPT-4	94.8	88.8	85.1	64.5	85.7	7.89
DEA-SQL	GPT-4	88.7	89.5	85.6	70.5	85.6	8.69
Self-augmented In-Context Learning							
SAFE-SQL	GPT-4	93.2	88.9	85.8	74.7	86.8	21.41
SAFE-SQL	GPT-4o	93.4	89.3	**88.4**	75.8	87.9	14.92
SAFE-SQL	GPT-4o-mini	93.6	87.5	86.1	75.2	87.4	15.33
SAFE-SQL	Llama3.1-70B-Instruct	90.4	88.2	86.2	78.2	86.8	23.52
SAFE-SQL	Llama3.3-70B-Instruct	92.0	80.5	81.0	62.9	80.5	22.46
SAFE-SQL	Qwen2.5-72B	87.6	74.5	77.0	52.4	74.5	28.51
SAFE-SQL	Gemma3-12B	92.4	90.7	85.1	**78.8**	88.2	13.28
SAFE-SQL	Gemma3-27B	93.6	89.8	87.4	**78.8**	**88.5**	14.26

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.

4.4 Performance Comparison

Spider Dataset 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. These multiple factors play a crucial role in enabling the model to generate more accurate and structurally sound SQL queries, especially

Method	Model	Execution Accuracy
Supervised Fine-Tuning (SFT)		
Syn-SQL	Sense13B	63.4
SQL-Palm	Palm	53.6
Zero-shot Methods		
Baseline	GPT-4	49.2
Baseline	GPT-4o	51.8
Baseline	GPT-4o-mini	51.2
C3-SQL	GPT-4	53.8
Few-shot Methods		
Din-SQL	GPT-4	55.9
Dail-SQL	GPT-4	55.4
ACT-SQL	GPT-4	52.8
PTD-SQL	GPT-4	57.0
DEA-SQL	GPT-4	52.4
Self-augmented In-Context Learning		
SAFE-SQL	GPT-4	58.9
SAFE-SQL	GPT-4o	**63.5**
SAFE-SQL	GPT-4o-mini	62.1
SAFE-SQL	Llama3.1-70B-Instruct	60.9
SAFE-SQL	Llama3.3-70B-Instruct	61.2
SAFE-SQL	Qwen2.5-72B	56.2
SAFE-SQL	Gemma3-12B	60.8
SAFE-SQL	Gemma3-27B	61.5

Table 2: Execution accuracy on Bird dataset.

Models	EX	EM
SAFE-SQL - GPT-4o	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 3: Ablation study results for SAFE-SQL, where removing each component leads to a performance drop.

in challenging scenarios where other approaches struggle. Notably, SAFE-SQL using open-source models such as Gemma3-27B outperforms high-cost methods based on GPT-4, highlighting its cost-effectiveness and strong capability.

Bird Dataset We also conduct experiments on the Bird Dev dataset in addition to the Spider dataset. Similar to Spider, SAFE-SQL consistently outperforms zero-shot and few-shot methods, achieving 63.5% execution accuracy with GPT-4o, which is even higher than Syn-SQL (63.4%), a supervised fine-tuning approach. This highlights the effectiveness of our SAFE-SQL using a self-augmented in-context learning method.

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 3. 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, empirically validating our architectural and design choices.

4.6 Analysis

Inference Time per Query on Spider Dev Set As shown in Table 1, we compare the inference time of SAFE-SQL with other methods. While baseline methods achieve faster inference via simple zero-shot prompts, they show lower accuracy. Few-shot methods are faster than SAFE-SQL but still underperform in execution accuracy. In contrast, SAFE-SQL leverages example augmentation and filtering process, achieving higher performance with a modest increase in inference time. Despite requiring three LLM calls, SAFE-SQL demonstrates strong zero-shot capabilities without relying on a training set, making the trade-off in latency worthwhile.

Score	$\cos\theta$	# 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 4: 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 4, 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

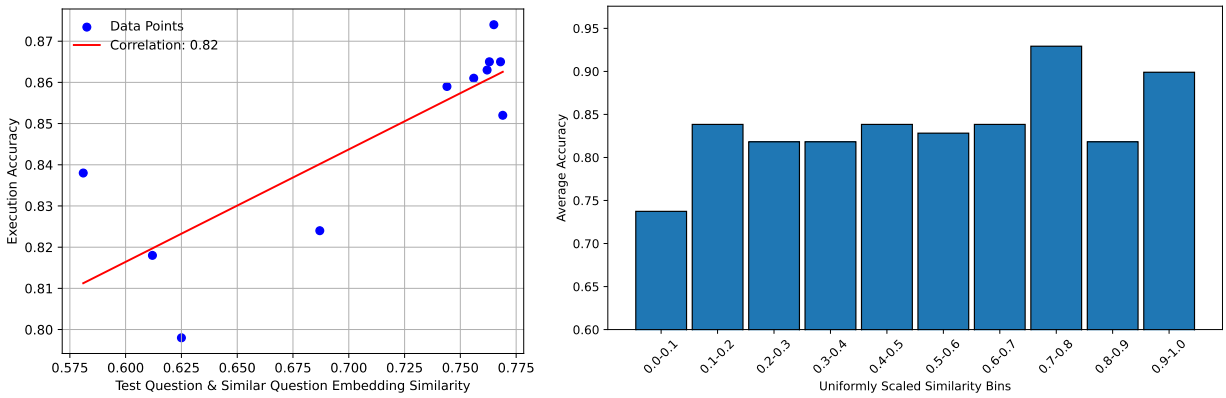

Figure 3: (Left) Correlation between question embedding similarity and average EX, (Right) Average EX across embedding similarity bins

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.

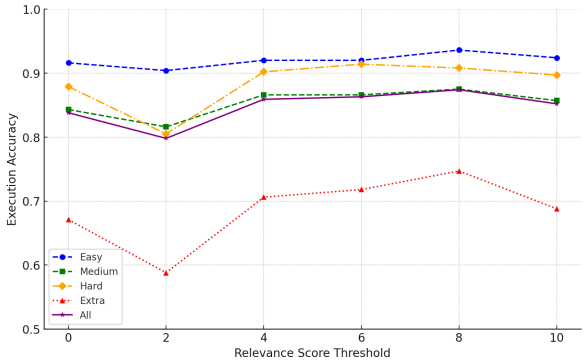

Figure 4: Performance of GPT-4o at different relevance score thresholds.

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 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 have six examples with scores ≥ 8, while others have three). The selected examples are then used during the final inference stage to generate SQL queries. The ≥ 8 threshold consistently produces the best results, validating the robustness of SAFE-SQL's relevance score filtering. The results demonstrate that selecting high-quality examples plays a critical

GOLD Question	GOLD SQL Query	Generated Question	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 5: Comparative examples of gold-questions and GPT-4o-generated SQL questions, including detailed reasoning paths and the process of computing relevance scores.

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

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), structural similarity (β), and reasoning path quality (γ)—on EX, we conduct experiments by varying their respective weightings. The results, presented in Table 6, 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 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 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 5, 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

We introduce SAFE-SQL, a novel unsupervised framework designed for Text-to-SQL. SAFE-SQL generates and filters high-quality self-augmented examples for in-context learning. Extensive experiments demonstrated that both the fine-grained example generation process and optimal threshold filtering contribute significantly to performance gains. Our method achieves state-of-the-art results, showing notable improvements over ablated versions and excelling particularly in challenging extra hard and unseen scenarios.

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

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.

Acknowledgement

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [RS-2021-II211341, Artificial Intelligence Graduate School Program (Chung-Ang University)] and a project for Collabo R&D between Industry, University, and Research Institute funded by Korea Ministry of SMEs and Startups in 2025.(RS-2025-02323112).

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*.

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.

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.

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*.

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.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Preprint*, arXiv:2305.03111.

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.

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

KaShun Shum, Shizhe Diao, and Tong Zhang. 2023. Automatic prompt augmentation and selection with chain-of-thought from labeled data. *arXiv preprint arXiv:2302.12822*.

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.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Rishit Toteja, Arindam Sarkar, and Prakash Mandayam Comar. 2025. In-context reinforcement learning with retrieval-augmented generation for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 10390–10397.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2021. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *Preprint*, arXiv:1911.04942.

Xubin Wang, Jianfei Wu, Yichen Yuan, Mingzhe Li, Deyu Cai, and Weijia Jia. 2024. Demonstration selection for in-context learning via reinforcement learning. *Preprint*, arXiv:2412.03966.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. *Preprint*, arXiv:2212.10560.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

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.

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 7.

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
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 7: 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 8.

B.3 Prompt for final inference.

For final inference, we use zero shot prompt as shown in figure 9.

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
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.

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
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 8: 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 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. 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 9: 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 10: Execution accuracy performance of different size of models of Qwen series across difficulty levels of spider dev set.

C Impact of model size

Performance based on generated examples across different model size As shown in Table 10, 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.

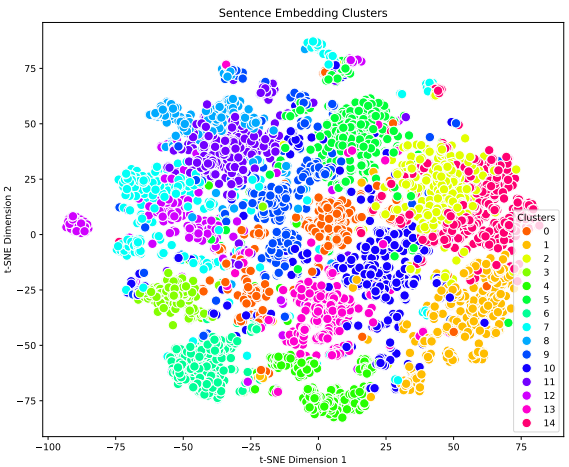

Figure 5: Embedding of spider dev set training questions.

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.