SQL-Guided Semantic Variation for Data Augmentation in Text-to-SQL

Isaque O. Nascimento[1], **Kelly R. Braghetto**[1]

[1] Instituto de Matemática, Estatística e Ciência da Computação
Universidade de São Paulo (USP) – São Paulo, SP – Brazil

isaque.nascimento@usp.br, kellyrb@ime.usp.br

Abstract. *Text-to-SQL maps natural language questions to SQL queries. LLMs have become a promising solution for this task, but question-SQL pairs are hard to produce. Data augmentation can expand training datasets, but many existing approaches focus on paraphrasing questions while preserving the original intent. This work proposes a SQL-guided augmentation technique that creates new examples through controlled semantic changes. The method modifies SQL components, such as columns, operators and values, while preserving query validity, and then aligns the natural language question with the transformed SQL using LLM support. The goal is to increase semantic diversity in Text-to-SQL datasets, while future work will evaluate its impact on model performance.*

1. Introduction

Relational databases remain the standard for structured data management, yet querying them requires technical SQL proficiency, limiting accessibility for non-technical users. Text-to-SQL bridges this gap by mapping natural language questions directly to executable SQL queries [Katsogiannis-Meimarakis and Koutrika 2023]. While Large Language Models (LLMs) have significantly advanced this domain, training robust architectures requires massive datasets. Annotated question-SQL pairs are highly expensive and labor-intensive to produce, particularly in low-resource or specialized domains [Katsogiannis-Meimarakis and Koutrika 2023]. Consequently, automated data augmentation is essential for expanding Text-to-SQL training sets without prohibitive manual annotation costs.

Data augmentation refers to techniques that generate additional training examples from existing ones, increasing the size or diversity of a dataset without fully manual annotation. In Text-to-SQL, some augmentation strategies include synonym substitution [Gan et al. 2021] and LLM-driven paraphrasing [Liu et al. 2024]. However, these techniques primarily operate on natural-language questions, usually preserving the original intent and SQL structure. This may help models handle different linguistic formulations of the same request, but it does not necessarily increase the semantic diversity of the underlying SQL queries.

In this work, we propose and discuss a SQL-guided data augmentation technique that goes beyond rephrasing. Given a question-SQL pair, our approach parses the SQL query into an Abstract Syntax Tree (AST), applies controlled transformations to SQL components such as columns, values, and operators, and regenerates a valid SQL query with modified semantics. Then, using LLM support, the natural language question is

aligned with the transformed SQL query, producing a new semantically varied Text-to-SQL example.

The goal of this work is to increase the semantic diversity of Text-to-SQL datasets while preserving SQL validity and reducing the need for fully manual annotation. The main contributions are: (i) a SQL-guided strategy for generating semantically varied Text-to-SQL examples; (ii) a transformation pipeline based on SQL AST manipulation; and (iii) an LLM-supported mechanism for aligning natural language questions with transformed SQL queries.

2. Theoretical Background

Text-to-SQL tasks involve translating a natural language (NL) question and an associated database schema into an executable SQL query that accurately reflects user intent [Katsogiannis-Meimarakis and Koutrika 2023]. A primary challenge in this domain is resolving linguistic ambiguity [Liu et al. 2025]. For instance, a natural request such as "Find the nearest hospitals to street ABC" lacks explicit constraints defining proximity thresholds. While deep learning models offer a promising mechanism for learning to resolve these contextual ambiguities, training robust architectures requires substantial, diverse datasets to generalize across heterogeneous query patterns [Liu et al. 2025].

2.1. Data Augmentation

Data augmentation is a strategy for generating synthetic examples from an original dataset, artificially increasing its size and diversity [Shorten et al. 2021]. In Text-to-SQL, this can be achieved by modifying the user input, generating linguistic variations or slight changes in intent, or by modifying the SQL query to create a new variation.

For example, consider the pair composed of the question *"How many schools have more than 200 students in Rio de Janeiro?"* and the corresponding SQL query:

```
SELECT COUNT(*) FROM schools
WHERE studentsCount > 200 AND city = 'Rio de Janeiro';
```

This pair can be transformed into a new one by changing only the natural language question, such as: ''*What is the total number of schools in Rio de Janeiro with over 200 students?"* while keeping the same SQL query. This creates a new example for the dataset and can be automated through LLMs, for instance [Liu et al. 2024].

2.2. SQL Abstract Syntax Trees

An Abstract Syntax Tree (AST) represents the syntactic structure of a query by decomposing the code into logical components such as clauses, predicates, column references, and operators. This structural representation enables algorithmic analysis and transformation beyond plain text manipulation. In this work, we use `SQLGlot`, a Python-based SQL parser, transpiler, and optimizer [Mao 2023], to parse SQL strings into ASTs and regenerate queries after mutation.

For our data augmentation pipeline, the AST allows the mutation algorithm to systematically identify valid modification targets. Rather than executing unsafe string searches, the pipeline inspects individual tree nodes to isolate elements such as equality predicates, numerical thresholds, range conditions, or aggregate functions. Once a target node is identified, the algorithm verifies it against the database schema to execute valid, context-aware mutations.

3. Proposed Data Augmentation Strategy

The data augmentation strategy proposed in this work aims to generate new Text-to-SQL examples by modifying the semantics of the original SQL query rather than applying only surface-level changes to the natural language question. Given a question-SQL pair and a custom database schema, the algorithm produces a transformed SQL query and a corresponding natural language question aligned with the new query semantics.

For example, Figure 1 illustrates a transformation in which both the aggregation function and the filter value are modified.

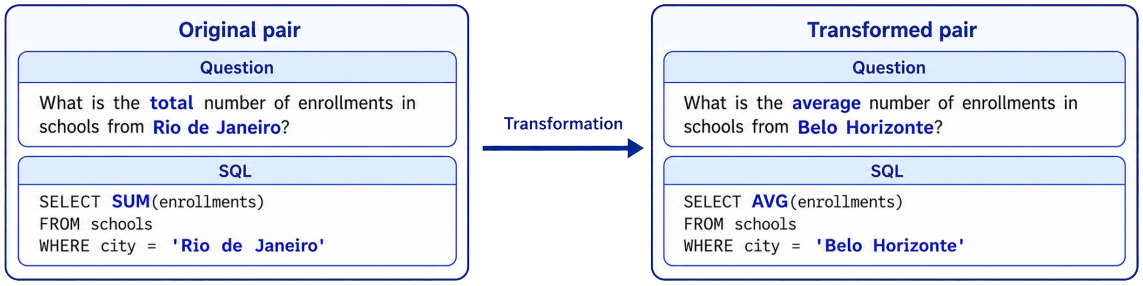

Figure 1. Example of a semantic transformation applied to a Text-to-SQL pair.

The algorithm first parses the input SQL query into an Abstract Syntax Tree (AST) using SQLGlot. It then traverses the tree and analyzes each node as a possible mutation target. For each node, a mutation function checks its type and surrounding context to determine whether a safe transformation can be applied. For instance, a literal value may be mutated only when it appears in a predicate involving a column described in the custom schema. This schema defines which tables and columns are eligible for mutation, as well as the constraints that guide how each column can be modified.

After applying one or more SQL mutations, the transformed query is converted back into SQL. Finally, the natural-language question is rewritten with LLM support to remain consistent with the modified query. The following subsections describe the custom schema used to constrain the transformations and the set of SQL mutations currently supported by the algorithm.

3.1. Custom Schema

The proposed method relies on a custom schema that defines which database elements can be safely modified during augmentation. This schema serves as a metadata layer over the original database schema, specifying the tables and columns eligible for mutation and the constraints that guide each transformation. By explicitly defining these constraints, the algorithm avoids arbitrary SQL modifications and reduces the risk of generating invalid or semantically inconsistent queries.

The custom schema is organized as a collection of table definitions. Each table entry contains its name and a list of column definitions. Column entries store both structural information, such as column name and type, and semantic information, including descriptions, valid value ranges, enum values, value groups, and semantic groups. These metadata fields are used by the mutation functions to decide whether a given AST node can be modified and which alternative values or columns may be selected.

3.1.1. Column Properties

Column properties define which mutations can be safely applied to each field and how the natural language question should be adapted after the SQL changes. Table 1 shows the main properties.

Table 1. Column metadata used to guide SQL mutations.

Property	Applies to	Purpose
`name`	All columns	Identifies the database column matched against SQL AST nodes.
`description`	All columns	Provides a human-readable meaning used in the mutation changelog.
`type`	All columns	Defines the mutation category, such as `number`, `date`, `string`, `enum`, or `binary`.
`min`, `max`	Numeric and date columns	Define valid bounds for threshold and range mutations.
`enums`	Enum columns	Lists the allowed values for categorical substitutions.
`value_group`	Enum columns	Groups enum values into higher-level sets that can be replaced together.
`value_group_labels`	Enum columns	Provides readable labels for value groups in the changelog.
`semantic_group`	Related columns	Restricts equivalent-column mutations to fields with related meanings.

3.2. Mutations

Mutations are controlled transformations applied to specific components of the SQL query. Table 2 shows the mutation types currently supported by the proposed method.

3.3. Natural Language Regeneration

After the SQL mutations are applied, the original natural language question must be rewritten to reflect the semantics of the transformed query. In the current implementation, this step is performed with Gemini 3.1 Flash-Lite. The model receives the original question-SQL pair, the transformed SQL query, and a mutation log describing the changes applied to the query, such as modified columns, operators, aggregations, or literal values.

The mutation log is used to constrain the rewriting process. Instead of requiring the LLM to infer the differences between the original and transformed SQL queries, the prompt explicitly indicates which semantic elements were changed. For enum mutations, the changelog also includes textual descriptions of the original and replacement enum values, allowing the LLM to use their domain meaning rather than relying only on database codes. The prompt instructs the model to adapt the original natural language question according to the new SQL query and to return only the rewritten question, ensuring that it remains fluent in the target language. This design reduces the risk of generating a question that is linguistically valid but inconsistent with the transformed SQL.

Table 2. Supported SQL mutations.

Mutation	Condition	Example
Binary	Equality predicate over binary columns	`has_library = 1` → `has_library = 0`
Enum equality	Equality predicate over enum columns	`state = 'RJ'` → `state = 'MG'`
Aggregate	Aggregate function in {`SUM`, `AVG`, `MIN`, `MAX`}	`SUM(x)` → `AVG(x)`
Threshold	Inequality over bounded numeric/date columns	`x > 100` → `x <= 350`
Between	Range predicate over bounded numeric/date columns	`x BETWEEN 100 AND 300` → `x BETWEEN 400 AND 900`
Equivalent column	Column with a configured semantic group	`SUM(enrollments)` → `SUM(teachers)`
Value group	`IN` predicate matching a configured enum group	`IN ('RJ','SP','MG','ES')` → `IN ('PR','SC','RS')`

4. Preliminary Intrinsic Evaluation

We conducted an intrinsic evaluation using 980 generated pairs from a real geospatial Text-to-SQL dataset [Lopes and Braghetto 2026] to analyze whether the pipeline produces controlled variations in both natural language (NL) questions and SQL queries. Downstream model evaluation remains future work. Table 3 provides qualitative examples of the main SQL-level mutations and corresponding NL regenerations.

Table 3. Qualitative examples of generated augmented pairs.

Original Question	Main Applied Mutations	Regenerated Question
In Minas Gerais, which schools with a computer lab are inside their municipality and within 2,500 m of the border with Rio de Janeiro?	State filter: MG → AM; border state: RJ → CE; facility: computer lab → uncovered sports court absent; distance: 2,500 → 4,339 m.	In Amazonas, which schools without an uncovered sports court are inside their municipality and within 4,339 m of the border with Ceará?
In Rio Grande do Sul, which state schools with renewable energy and installed TV are inside their municipality and within 4,000 m of the border with Santa Catarina?	State filter: RS → PE; border state: SC → DF; dependency: state → federal; infrastructure: renewable energy/TV → sewage/desktop computer; distance: 4,000 → 3,692 m.	In Pernambuco, which federal schools with sewage service available and at least one desktop computer for student use are inside their municipality and within 3,692 m of the border with the Federal District?

To quantify NL variation, we compute a semantic distance score defined as $1 - cosine\_similarity$ (clipped to $[0, 1]$) between the original and regenerated question embeddings. Higher scores denote stronger textual variation. For SQL queries, embedding distances proved uninformative because shared schema tokens and clauses artificially inflate similarity despite critical structural changes. Instead, we measure SQL variation using an AST component matching metric: the ratio of altered normalized AST components (e.g., selections, joins, predicates, spatial functions) to the total component count. This estimates structural divergence rather than execution equivalence.

Table 4 summarizes the intrinsic metrics. The mean NL variation (0.155) indicates moderate lexical and semantic diversity while preserving the underlying intent. The mean SQL component matching score (0.113, $\max = 0.533$) reflects clear structural differentiation. The most frequent mutations occurred within predicates, spatial functions, and selected expressions, confirming that the pipeline successfully alters semantically relevant elements while maintaining schema compatibility and query validity.

Table 4. Intrinsic evaluation metrics over 980 examples.

Metric	Average	Median	P90	Max
Question embedding variation	0.155	0.147	0.284	0.567
SQL component matching	0.113	0.096	0.226	0.533

5. Related Work

Data augmentation has been explored as a strategy to improve robustness and diversity in NLP datasets. Zhu et al. [Zhu et al. 2023] propose RACE, a counterfactual data augmentation method for multi-hop fact verification based on an Explain-Edit-Generate pipeline. Their method first identifies rationales, then edits evidence, and finally generates counterfactual claims, using checking and filtering steps to preserve logical consistency. Although RACE addresses fact verification rather than Text-to-SQL, it is related to our work because it underscores the importance of decomposing augmentation into controlled stages rather than relying solely on unconstrained text generation.

Cai et al. [Cai et al. 2025] propose TEXT2SQL-FLOW, a SQL-aware data augmentation framework that generates semantically valid and structurally diverse question-SQL pairs from limited seed data. Their pipeline augments SQL queries across multiple dimensions, verifies SQL query executability, generates natural-language questions, and filters NL-SQL correspondence. Our work is complementary, but more focused: instead of generating large-scale datasets through broad LLM-driven SQL synthesis, we apply explicit AST-based mutations over existing SQL queries, guided by a custom schema. This allows controlled semantic variation while preserving SQL validity, using LLM support only to align the natural-language question with the transformed query.

6. Conclusion

This work proposed a SQL-guided data augmentation strategy for Text-to-SQL datasets. Instead of only paraphrasing natural language questions, the method applies controlled mutations to SQL queries via AST manipulation and uses a custom schema to preserve query validity. The natural language question is then regenerated with LLM support to reflect the modified SQL semantics. The source code of the proposed algorithm is publicly available at: `https://github.com/ioolliver/ast-driven-data-augmentation-algorithm`.

By introducing changes in columns, operators, values, aggregation functions, and value groups, the approach aims to increase the semantic diversity of augmented examples. As future work, we plan to evaluate the generated pairs in terms of semantic variation, schema coverage, and question-SQL consistency, and later investigate their impact on downstream Text-to-SQL model performance.

Use of Generative AI

An AI tool (GPT 5.5) was used to generate Figure 1, review writing, and suggest improvements in structure, clarity, and academic tone. However, all technical ideas, methodological decisions, content, and references were defined, reviewed, and verified by the human authors, who remain fully responsible for the final version of this work.

Project Context

This work is part of an undergraduate research project focused on data augmentation strategies for low-resource domains. Isaque Nascimento was funded by a CNPq-PIBIC scholarship. This work was partially supported by the São Paulo Research Foundation (FAPESP) [grants #2023/00779-0 and #2023/18026-8]; and the National Council for Scientific and Technological Development (CNPq) [grant #420623/2023-0].

The research is motivated by the long-term goal of providing a natural language interface for public open data portals, empowering citizens to query data without any SQL knowledge. The data augmentation strategy proposed in this paper contributes to this direction by investigating how scarce Text-to-SQL datasets can be expanded in domain-specific scenarios, such as Brazilian sociodemographic databases.

References

Cai, Q., Liang, H., Xu, C., Xie, T., Zhang, W., and Cui, B. (2025). Text2sql-flow: A robust sql-aware data augmentation framework for text-to-sql. *arXiv:2511.10192*.

Gan, Y., Chen, X., Huang, Q., Purver, M., Woodward, J. R., Xie, J., and Huang, P. (2021). Towards robustness of text-to-sql models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and 11th International Joint Conference on Natural Language Processing*, pages 2505–2515.

Katsogiannis-Meimarakis, G. and Koutrika, G. (2023). A survey on deep learning approaches for text-to-SQL. *The VLDB Journal*, 32(4):905–936.

Liu, S., Almohaimeed, S., and Wang, L. (2024). Reformer: A chatgpt-driven data synthesis framework elevating text-to-sql models. In *2024 International Conference on Machine Learning and Applications (ICMLA)*, page 828–833. IEEE.

Liu, X., Shen, S., Li, B., Ma, P., Jiang, R., Zhang, Y., Fan, J., Li, G., Tang, N., and Luo, Y. (2025). A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*.

Lopes, D. O. and Braghetto, K. R. (2026). AtlasSQL-BR. `https://huggingface.co/datasets/datafromlopes/atlas-sql-br`. Accessed: 2026-05-18.

Mao, T. (2023). Sqlglot documentation. `https://sqlglot.com/sqlglot.html`. Accessed: 2026-05-18.

Shorten, C., Khoshgoftaar, T. M., and Furht, B. (2021). Text data augmentation for deep learning. *Journal of Big Data*, 8(1):101.

Zhu, Y., Si, J., Zhao, Y., Zhu, H., Zhou, D., and He, Y. (2023). Explain, edit, generate: rationale-sensitive counterfactual data augmentation for multi-hop fact verification. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13377–13392.