A Demonstration of SQLyzr: A Platform for Fine-Grained Text-to-SQL Evaluation and Analysis

Sepideh Abedini
University of Waterloo
sepideh.abedini@uwaterloo.ca

M. Tamer Özsu
University of Waterloo
tamer.ozsu@uwaterloo.ca

ABSTRACT

Text-to-SQL models have significantly improved with the adoption of Large Language Models (LLMs), leading to their increasing use in real-world applications. Although many benchmarks exist for evaluating the performance of text-to-SQL models, they often rely on a single aggregate score, lack evaluation under realistic settings, and provide limited insight into model behaviour across different query types. In this work, we present SQLyzr, a comprehensive benchmark and evaluation platform for text-to-SQL models. SQLyzr incorporates a diverse set of evaluation metrics that capture multiple aspects of generated queries, while enabling more realistic evaluation through workload alignment with real-world SQL usage patterns and database scaling. It further supports fine-grained query classification, error analysis, and workload augmentation, allowing users to better diagnose and improve text-to-SQL models. This demonstration showcases these capabilities through an interactive experience. Through SQLyzr's graphical interface, users can customize evaluation settings, analyze fine-grained reports, and explore additional features of the platform. We envision that SQLyzr facilitates the evaluation and iterative improvement of text-to-SQL models by addressing key limitations of existing benchmarks. The source code of SQLyzr is available at https://github.com/sepideh-abedini/SQLyzr.

1 INTRODUCTION

Relational database management systems (RDBMSs) are widely used to store and manage structured data across many domains, such as healthcare, finance, and enterprise systems [15]. Accessing this data typically requires knowledge of SQL, which can be challenging for non-expert users. Text-to-SQL models aim to address this challenge by translating natural language utterances into executable SQL queries [10], enabling intuitive access to structured data. Reducing barriers to accessing RDBMSs using natural language has long been a goal, going back to early work by Codd [4], followed by many subsequent efforts, e.g., [6, 12, 16, 18].

Current state-of-the-art approaches rely on large language models (LLMs) to generate SQL queries from natural language questions [7]. Recent advances in LLMs have significantly improved the performance of text-to-SQL models. However, existing benchmarks have not kept pace with these advancements and suffer from several limitations that restrict their effectiveness for evaluating text-to-SQL models in real-world deployment settings.

First, existing benchmarks typically report only a single aggregate correctness score, which does not reveal which query types are more challenging for a model. Moreover, relying solely on correctness overlooks other important aspects of the generated queries, such as execution efficiency and structural complexity, which are critical factors for deploying text-to-SQL models in production environments.

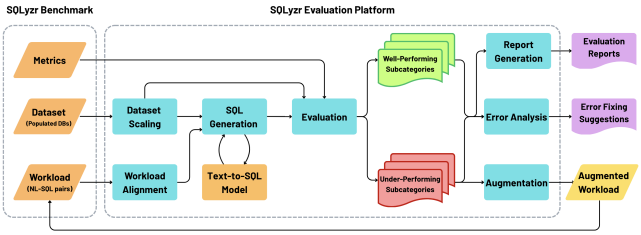


Figure 1: Overview of SQLyzr

Second, these benchmarks rely on fixed and small-scale databases. While this simplifies the evaluation, it does not capture how generated queries behave under realistic, large-scale settings, particularly in terms of efficiency.

Third, these benchmarks often use workloads that do not reflect real-world SQL usage patterns, limiting their ability to reliably predict model performance in practical deployments.

Finally, traditional benchmarks are inherently static. Once a model achieves a relatively high score, subsequent evaluations provide limited diagnostic value. As a result, developers need to construct new test cases or switch benchmarks to identify remaining weaknesses. Consequently, existing benchmarks are often used as static, one-time evaluation tools, and are not well suited for integration into iterative model development workflows.

In this work, we demonstrate SQLyzr [1], a comprehensive benchmark and evaluation platform for text-to-SQL models designed to address these limitations. SQLyzr defines a benchmark specification consisting of a workload, a dataset, and a set of evaluation metrics. In addition, it provides a configurable benchmarking platform that supports flexible evaluation settings and enables fine-grained analysis, workload augmentation, and evaluation under scaled database settings. While the benchmark specification and evaluation framework are described in [1], this work focuses on the practical usage and analysis capabilities of the platform through an interactive demonstration, where users can configure evaluations and explore SQLyzr's key features via its graphical user interface (GUI).

2 SQLYZR OVERVIEW

SQLyzr consists of two complementary components: (i) a benchmark specification that defines a workload, a dataset, and a set of evaluation metrics (Section 2.1), and (ii) a benchmarking platform that enables configurable evaluation of one or more text-to-SQL models (Section 2.2).

2.1 SQLyzr Benchmark Specification

The SQLyzr benchmark consists of three main components.

Workload. The workload is a collection of natural language questions paired with SQL queries used to evaluate text-to-SQL models. The SQL queries are treated as ground truth and are manually annotated to ensure correctness.

SQLyzr introduces a comprehensive taxonomy of SQL queries based on their structural characteristics, such as nesting patterns or the presence of specific SQL constructs. This taxonomy classifies queries into six categories and 36 subcategories, ordered by increasing complexity, reflecting the progression of SQL concepts in standard database textbooks [14]. This classification enables fine-grained evaluation by reporting scores at both category and subcategory levels, allowing users to identify which types of queries are more challenging for a text-to-SQL model. It also supports the computation of complexity-based metrics and targeted workload augmentation, as described in Section 2.2.

The workload includes 20,979 data points, with approximately 11% allocated for training and the remainder for evaluation. The training split is used to provide demonstration examples for in-context learning [2], while the evaluation split is used exclusively for benchmarking. The SQLyzr workload is constructed from existing Text-to-SQL benchmarks, including Spider [17], BIRD [9], and BEAVER [3], capturing a diverse set of query types and structural complexities.

SQLyzr also provides a subset of the workload aligned with the empirical query distribution in SQLShare [8], enabling evaluation that better reflects real-world SQL usage patterns. The query distribution is defined based on the frequency of queries across categories in the SQL taxonomy.

Dataset. The dataset consists of populated databases. Each data point in the workload is associated with a database on which both generated and ground truth queries are executed, and their results are compared to measure correctness and efficiency. The SQLyzr dataset is composed of databases from Spider, BIRD, and BEAVER, comprising 286 databases across SQLite and MySQL RDBMSs.

Evaluation Metrics. SQLyzr uses a set of evaluation metrics to assess text-to-SQL models that go beyond correctness, which is typically used as the sole metric in existing benchmarks. In addition to correctness, SQLyzr captures other important aspects of generated queries, such as efficiency, structural complexity, and generation cost.

Execution Accuracy (EA): Measures correctness by comparing the execution results of generated and ground truth queries on a single database instance.

Exact Match (EM): Compares the structural components of generated and ground truth queries following the Spider benchmark definition [17]. To support complex queries, we implemented an AST-based comparison that is more robust than the string-based approach originally provided in the Spider benchmark[1], which is limited to Spider's relatively simple queries.

Complexity Consistency (CC): Measures whether the generated query introduces unnecessary structural complexity compared to the ground truth query, e.g., through an unnecessary join. A query is considered consistent if its category is not more complex than the category of the ground truth query.

[1]https://github.com/taoyds/spider

Execution Time Consistency (ETC): Evaluates whether the generated query executes within an acceptable relative time overhead compared to the ground truth query, capturing the execution efficiency of the generated query.

Token Usage (TU): Measures the number of tokens used by the language model to process input and generate the query. Since we focus on LLM-based text-to-SQL models, this metric provides an estimate of generation cost.

2.2 SQLyzr Evaluation Platform & Methodology

SQLyzr provides a configurable platform and methodology for evaluating text-to-SQL models. An overview of SQLyzr is illustrated in Figure 1. The evaluation begins by selecting a workload, a dataset, and one or more text-to-SQL models, along with a subset of evaluation metrics defined in Section 2.1. SQLyzr then executes a multi-stage pipeline that produces detailed evaluation reports and diagnostic insights.

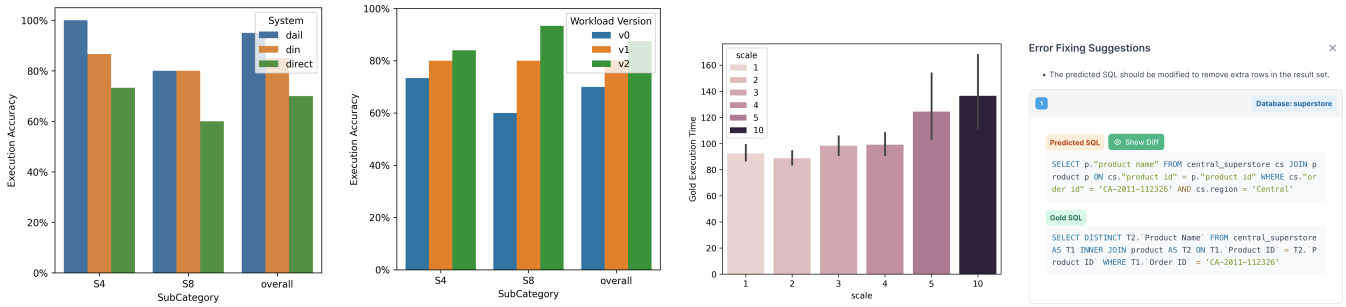
Dataset Scaling. As an optional preprocessing step, SQLyzr supports database scaling by generating synthetic data, enabling evaluation under large-scale conditions. SQLyzr leverages the SDV [11] framework to train a generative model on each database and synthesize additional rows that preserve the statistical properties of the original data. These rows are then inserted into the databases to scale them to the desired sizes.

Workload Alignment. As another preprocessing stage, SQLyzr aligns the workload with a target query distribution. Alignment with empirical data, such as SQLShare, enables evaluations that better reflect real-world SQL usage patterns. However, SQLyzr is not limited to SQLShare, and users can align workloads with any target distribution. While this demonstration includes a specific instance of this functionality, full support for aligning workloads with custom distributions is available in the codebase.

SQL Generation. As the first evaluation stage, SQLyzr invokes the selected text-to-SQL model to generate a SQL query for each natural language question in the workload. The model is treated as a black-box component that receives a question and the corresponding database schema and produces a SQL query.

Evaluation. In the next stage, generated queries are evaluated against the ground truth queries using the specified metrics. This evaluation captures multiple aspects of model performance, including not only correctness but also efficiency, structural complexity, and generation cost. To support fine-grained analysis, SQLyzr assigns each query to a category and subcategory based on the defined SQL taxonomy. Category assignment is implemented using abstract syntax tree (AST) traversal, which enables the identification of complex patterns in the queries.

Analysis and Reporting. Following evaluation, SQLyzr produces reports at multiple levels of granularity. Results are presented both as aggregate scores and at the level of categories and subcategories, enabling detailed comparison across models. Figure 2a shows an example execution accuracy plot, where both overall and subcategory-level scores are reported. For each metric, SQLyzr also generates workload comparison plots that capture the effect of iterative augmentation (Figure 2b) and scaling plots that illustrate how scores change as database size increases (Figure 2c).



(a) Model comparison **(b) Workload comparison (fixed model)** **(c) Dataset scaling** **(d) Error fixing suggestions**

Figure 2: Example evaluation plots and error analysis results produced by SQLyzr

Error Analysis. This stage provides analysis to support the diagnosis of incorrectly generated queries. Due to the ambiguity of natural language questions, a single question may have multiple valid SQL representations. Therefore, execution accuracy can be overly strict for measuring correctness, as even minor deviations from the ground truth query, such as different column ordering, may cause a generated query to be marked as incorrect. In this stage, SQLyzr identifies a subset of such cases by applying a set of transformations to generated queries and their execution results. If a transformed version of a generated query produces the same result as the ground truth query, the corresponding transformations are reported as potential fixes. Figure 2d shows an example of such repair suggestions.

Workload Augmentation. This stage is performed after an initial round of evaluation. Based on the reported performance, SQLyzr identifies underperforming subcategories using a user-specified score threshold and generates additional question-SQL pairs within those subcategories, targeting model weaknesses. These new data points are added to the workload, emphasizing challenging query types and supporting iterative model improvement. This process enables repeated re-evaluation of improved models and helps uncover new weaknesses, allowing SQLyzr to function as an adaptive test suite rather than a one-time benchmark. Figure 2b shows an example workload comparison plot illustrating the effect of augmentation across workload versions.

2.2.1 Implementation Details of SQLyzr. This section describes implementation details and design choices that enhance the usability and efficiency of SQLyzr for benchmarking.

Batch Mode. SQLyzr supports both synchronous and asynchronous batch API calls for interacting with LLMs. While synchronous mode enables fast experimentation with immediate feedback, batch mode supports large-scale evaluations and reduces costs by up to 50%. SQLyzr provides a unified interface for LLM interaction, allowing users to switch between modes at runtime.

Optimizations. SQLyzr leverages parallelization, asynchronous execution, and caching to accelerate evaluations. Users can also adjust runtime parameters to customize these optimizations and maximize their effectiveness.

Configurability and Modularity. SQLyzr provides various configuration options for customizing the evaluation. In addition to selecting the text-to-SQL models, users can specify parameters such as the number of evaluation iterations, the LLM used for SQL generation, and its temperature. Moreover, the platform is designed with a modular architecture that supports extensions without modifying core components. User-defined text-to-SQL models can be integrated by implementing a simple software interface. Users can also incorporate custom workloads and datasets beyond those already specified in the benchmark (Section 2.1).

Usability. SQLyzr provides both a graphical user interface (GUI) and a command-line interface (CLI) for convenient interaction with the platform. This demonstration presents the SQLyzr GUI.

Database Engine Compatibility. The current version of SQLyzr supports SQLite and MySQL engines. However, SQLyzr is not limited to these engines, and users can easily extend the support to additional engines or dialects by implementing a simple driver.

3 INTERACTIVE DEMONSTRATION

This demonstration provides interactive scenarios for evaluating text-to-SQL models using SQLyzr and highlights key capabilities of the platform, including fine-grained evaluation, workload augmentation, and dataset scaling.

For demonstration purposes, we provide three small samples of SQLyzr's workload with increasing difficulty levels, each containing 20 data points. We include two state-of-the-art text-to-SQL models, DIN-SQL [13] and DAIL-SQL [5], along with a simple baseline, *Direct-LLM*, which generates SQL queries from natural language questions using a simple LLM prompt. The LLM used for query generation can be selected by the user through the GUI.

The demonstration begins with users configuring the evaluation through the main *Dashboard* of SQLyzr (Figure 3). Users then start the evaluation by clicking the *Run SQLyzr* button. Upon execution, SQLyzr produces a set of plots that visualize fine-grained evaluation results, enabling detailed analysis and comparison.

Scenario 1: Models Comparison and Diagnosis. In this scenario, users evaluate and compare multiple models using one of the sample workloads. After selecting models and configuring parameters such as the number of iterations, the LLM choice, and the temperature, users start the evaluation and monitor its progress through the *Logs* panel. Upon completion, the *Plots* panel presents results at both aggregate and fine-grained levels. These fine-grained plots reveal differences in model behaviour that are not visible through aggregate metrics alone, allowing users to identify model strengths

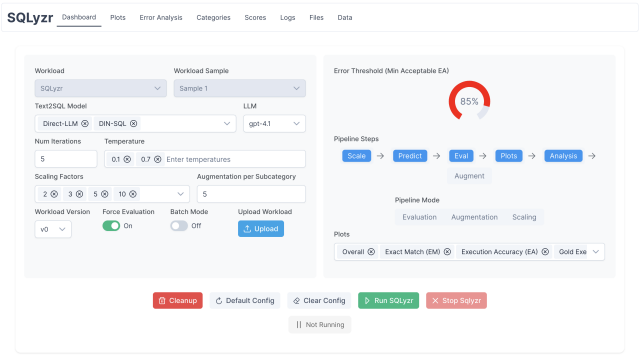


Figure 3: SQLyzr *Dashboard* for configuring evaluation and controlling pipeline execution

and weaknesses across query types (Figure 2a). The *Error Analysis* panel further highlights incorrect but fixable queries and suggests potential fixes, helping users to understand the causes of model errors and diagnose failures more effectively (Figure 2d).

Scenario 2: Iterative Workload Augmentation. This scenario demonstrates how SQLyzr supports adaptive evaluation through workload augmentation. Starting from an initial workload, users identify underperforming subcategories based on the evaluation results and specify a score threshold to guide augmentation. SQLyzr then generates new data points targeting these subcategories and extends the workload accordingly. By repeating this process, users construct progressively refined workloads that emphasize more challenging queries for the evaluated model. Users can then inspect evaluation plots that compare metrics across different versions of the augmented workloads (Figure 2b).

Scenario 3: Evaluation under Dataset Scaling. The final scenario demonstrates SQLyzr's dataset scaling capability using the same workload from the augmentation scenario. Users begin by selecting desired *Scaling Factors* through the *Dashboard* and executing the evaluation pipeline via the *Run SQLyzr* button. In the first stage, synthetic data is generated and inserted into the databases, after which SQLyzr evaluates the model on the scaled data. Following execution, SQLyzr produces plots showing how evaluation scores evolve as database size increases (Figure 2c). This enables users to assess model performance under conditions that more closely resemble real-world settings.

4 CONCLUSION

In this work, we present SQLyzr, a comprehensive benchmark and evaluation platform for text-to-SQL models. Through an interactive demonstration, users explore SQLyzr's capabilities via its GUI. We highlight several features of SQLyzr that address key limitations of existing benchmarks, including fine-grained evaluation, workload augmentation, and dataset scaling. We envision that SQLyzr facilitates the evaluation and iterative refinement of text-to-SQL models by moving beyond static benchmarking toward an adaptive and development-oriented evaluation framework.

REFERENCES

[1] Sepideh Abedini. 2026. *SQLyzr: A Comprehensive Benchmark and Framework for Evaluating Text-to-SQL Systems.* Master's thesis. University of Waterloo. https://hdl.handle.net/10012/23045

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.

[3] Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Moe Kayali, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2024. BEAVER: An Enterprise Benchmark for Text-to-SQL. arXiv 2409.02038. https://doi.org/10.48550/ARXIV.2409.02038

[4] E. F. Codd. 1974. Seven Steps to Rendezvous with the Casual User. In *In Proc. IFIP Working Conf. Database Management*, J. W. Klimbie and K. L. Koffeman (Eds.). 179–200.

[5] 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. arXiv 2308.15363. https://doi.org/10.48550/ARXIV.2308.15363

[6] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. 1978. Developing a natural language interface to complex data. 3, 2 (1978), 105–147. https://doi.org/10.1145/320251.320253

[7] 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. arXiv 2406.08426. https://doi.org/10.48550/ARXIV.2406.08426

[8] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. 2016. SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment. 281–293.

[9] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. *Advances in Neural Information Processing Systems* 36 (2024).

[10] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 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* 37, 4 (2025), 1954–1972. https://doi.org/10.1109/TKDE.2024.3496929

[11] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. 2016. The Synthetic Data Vault. In *Proc. IEEE International Conference on Data Science and Advanced Analytics*. IEEE, 399–410. https://doi.org/10.1109/DSAA.2016.49

[12] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. Association for Computing Machinery, New York, NY, USA, 149–157. https://doi.org/10.1145/604045.604070

[13] Mohammadreza Pourreza and Davood Rafiei. 2024. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Proc. Advances in Neural Information Processing Systems*, Vol. 36. 30557–30584.

[14] A. Silberschatz, H. Korth, and S. Sudarshan. 2019. *Database System Concepts* (7 ed.).

[15] Michael Stonebraker and Andrew Pavlo. 2024. What Goes Around Comes Around... And Around... *ACM SIGMOD Record* 53, 2 (2024), 21–37. https://doi.org/10.1145/3673562.3673568

[16] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. 7567–7578. https://aclanthology.org/2020.acl-main.677.pdf

[17] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proc. 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921.

[18] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. arXiv 1709.00103. https://arxiv.org/abs/1709.00103