European Research Studies Journal
Volume XXVII, Special Issue 3, 2024

pp. 426-438

Optimizing Customer Support Using Text2SQL to Query Natural Language Databases

Submitted 13/09/24, 1st revision 22/09/24, 2nd revision 07/10/24, accepted 30/10/24

Michał Maj[1], Damian Pliszczuk[2], Patryk Marek[3], Weronika Wilczewska[4], Bartosz Przysucha[5], Tomasz Rymarczyk[6]

Abstract:

Purpose: *This paper explores the challenges and potential solutions associated with integrating Text2SQL technology into customer support operations. By leveraging large language models (LLMs) and tools like Vanna.AI, the study aims to enhance the efficiency and accuracy of handling customer queries without requiring specialized SQL knowledge.*
Design/Methodology/Approach: *A comprehensive analysis was conducted comparing the effectiveness of three large language models—Llama3:70b-instruct, Gemma2:27b, and Codegemma—in generating correct SQL queries from natural language questions. The models were trained with identical datasets and evaluated using six benchmark questions over two iterations, with and without detailed database schema information. Performance metrics included correctness of the generated queries and response times.*
Findings: *The results indicated that while Llama3 and Gemma2 initially demonstrated higher accuracy, the addition of detailed database schema information did not improve model performance. Instead, it led to decreased accuracy and increased response times, particularly for Llama3. Codegemma showed shorter response times but slightly lower accuracy. The study highlights that excessive contextual information can overwhelm LLMs, suggesting the need for optimized context provision.*
Practical Implications: *The findings suggest that simplifying database schema information and focusing on essential contextual data can enhance the performance of LLMs in generating SQL queries. Implementing tools like Vanna.AI, which utilize Retrieval Augmented Generation (RAG), can improve customer support processes by enabling quick and accurate data access without specialized SQL expertise.*
Originality/Value: *This paper provides valuable insights into the practical challenges of implementing Text2SQL technology in customer support. It offers recommendations for balancing context provision and model capabilities, contributing to the optimization of LLM performance in real-world applications.*

Keywords: *Text2SQL; Customer Support; Large Language Models; Vanna.AI; Retrieval Augmented Generation; SQL Query Generation.*

[1]*WSEI University in Lublin, Poland, michal.maj@wsei.pl;*
[2]*Netrix S.A., Poland, WSEI University in Lublin, Poland, damian.pliszczuk@netrix.com.pl;*
[3]*Netrix S.A., Poland, patryk.marek@netrix.com.pl;*
[4]*Lublin University of Technology, Lublin, Poland, w.wilczewska@pollub.pl;*
[5]*Lublin University of Technology, Lublin, Poland, b.przysucha@pollub.pl;*
[6]*Netrix S.A., Poland, WSEI University in Lublin, Poland, tomasz.rymarczyk@netrix.com.pl;*

JEL Codes: *C45, C61, M31, L8, D83.*

Paper Type: *Research article.*

1. Introduction

In today's dynamic business environment, effective customer service is crucial to maintaining customer satisfaction and loyalty. Companies are striving to streamline their support processes, looking for innovative technologies that can bring human interactions closer to complex data systems. One groundbreaking approach in this regard is the integration of Text2SQL technology into customer service operations.

Text2SQL technology represents a significant advance in database querying, enabling support personnel without SQL expertise to access critical information by asking questions in natural language. This has the potential to revolutionize the way customer service teams interact with vast repositories of data, which can lead to faster response times, more accurate information retrieval and, ultimately, an improved customer experience (Praveen *et al.,* 2024; Zhang *et al.,* 2024).

However, the introduction of Text2SQL in a customer service context presents a number of research challenges. First, natural language understanding (NLU) presents significant difficulties. Interpreting the nuances and context of customer queries is a complex task that requires advanced NLU capabilities. Studies by Praveen *et al.* (2024) and Zhang *et al.* (2024) highlight the difficulty in accurately capturing the subtle meanings and intentions behind a variety of customer queries, especially when they involve colloquial language or implicit context.

In addition, dealing with domain-specific terminology and jargon presents another challenge. Dima *et al.* (2021) point out that customer service systems must be able to recognize and correctly interpret industry-specific terms, acronyms and technical language that may not be part of the general vocabulary.

This is particularly important in specialized fields such as healthcare, finance or technology, where misinterpretation can lead to incorrect query results and potentially mislead customers. Another challenge is the complexity of queries. Translating complex, multi-part queries into accurate SQL queries is a difficult task, requiring advanced natural language processing and query generation techniques.

Beurer-Kellner *et al.* (2023) show that customer queries often involve many related aspects that must be correctly analyzed and transformed into a coherent SQL structure. Dealing with nested queries and complex data join operations adds another level of complexity. Systems must be able to construct complex SQL statements that

accurately reflect the relationships between data from different tables.

Understanding the database schema presents an additional difficulty. Mapping natural language elements to the appropriate tables and columns is a crucial and complex task, requiring advanced algorithms and machine learning techniques. Fu *et al.* (2023) emphasize that this process involves accurately interpreting the semantic meaning of customer queries and correlating them with the appropriate database structures.

Dealing with differences in database designs across organizations further complicates the process. Biswal et al. (no date) point to the need for flexible and robust algorithms that can quickly learn and adapt to new database structures without compromising on accuracy or performance.

Resolving ambiguity is another important aspect. Clarifying vague or imprecise customer queries requires sophisticated natural language processing and contextual understanding. Mahmood *et al.* (2024) show that queries often contain ambiguous language, incomplete information or multiple possible interpretations. The challenge is to develop systems that can identify these ambiguities and implement effective disambiguation techniques, such as generating follow-up questions or using context from previous interactions.

A significant challenge in developing Text2SQL systems is multilingual support. Developing systems that support queries in multiple languages is a complex task. Liu *et al.* (no date) and Yang *et al.* (2024) emphasize that it requires not only translating queries, but also understanding the linguistic nuances and structural differences between languages. Ensuring consistent performance across languages requires advanced NLP models capable of adapting to unique grammatical and cultural structures.

Privacy and security are key in the implementation of Text2SQL systems. Ensuring the protection of sensitive data while providing useful answers is a challenge that requires a balance between information accessibility and data confidentiality. Yan *et al.* (2024) emphasize the need for robust security mechanisms such as data masking, encryption and advanced access control algorithms. Scalability, too, is a critical aspect of deploying Text2SQL systems in high-traffic environments.

Zulfikar *et al.* (2024) show that handling large volumes of simultaneous queries in real time requires advanced query processing techniques, load balancing and the use of distributed computing resources. Optimizing performance for large and complex databases also requires advanced indexing and caching strategies.

Error handling and feedback are crucial for efficient and user-friendly systems. Providing meaningful error messages and suggestions helps maintain user confidence and facilitates troubleshooting. Lin (2023) emphasizes that systems

should offer intelligent error handling mechanisms that communicate problems in a clear and understandable way while suggesting possible solutions. Integration with existing systems is critical to implementation success.

Tomova *et al.* (2024) point out that Text2SQL systems must be compatible with a variety of customer support platforms, CRM and database management systems. This requires a flexible architecture and APIs that allow seamless integration without disrupting current processes. Dealing with temporal and contextual queries presents unique challenges.

Li *et al.* (2024) emphasize that systems must accurately interpret and execute queries related to temporal data, taking into account different temporal expressions and temporal logic. Maintaining context between related queries is crucial to the consistency of customer interactions.

Adapting to changing data structures is essential to the long-term effectiveness of Text2SQL systems. Biswal et al. (no date) emphasize that systems must be able to adapt to changes in the database schema without extensive re-training, which requires advanced machine learning techniques.

Finally, transparency and understandability of system operation are key to building trust and effective use of technology. Song et al. (2024) emphasize that providing clear explanations of query interpretation and execution allows support staff to understand and verify system operations, which is essential for maintaining data integrity and operational efficiency.

2. Use of large Language Models

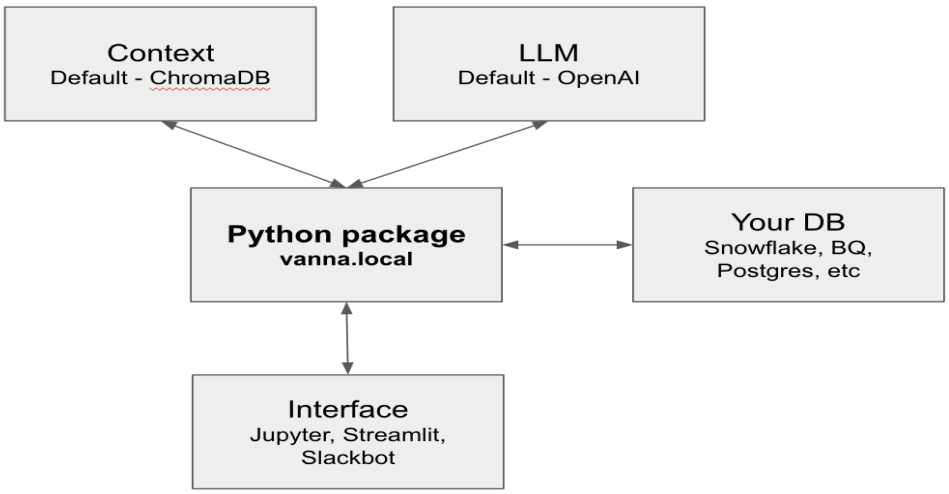
Vanna.AI is a set of tools for generating SQL queries using large language models (LLMs). Unlike the classic approach, where models are already learned and generate queries based on that, the essence of working with Vanna.AI is the process of learning the model so that it returns queries that best match the specifics of the data contained in the database to which it is connected. The tool uses the Retrieval Augmented Generation (RAG) technique to improve the quality of SQL queries generated by language models connected to any database.

Vanna.AI is characterized by several key features. First of all, it is open-source, which means it is available to a wide range of users and can be freely modified. It uses the RAG technique to select the most relevant data as the context for the model, which significantly increases the accuracy of the generated queries.

The tool allows connection to any database system and the use of a vector database to store training data, with a choice of different solutions, including open-source ones such as ChromaDB. In addition, Vanna.AI allows connection to any language models, including Ollama, a local, free and open-source tool with a wide selection of

available models, such as Llama 3 or Gemma 2. The tool also offers the possibility to continuously train the RAG layer by saving correctly generated queries as training data, and allows you to build your own user interface or use one of three prepared solutions: Slack, Flask or Streamlit.

Figure 1. *Functional diagram of the Vanna.AI packageanna.AI*



Source: *Own study.*

Adding RAG functionality to the flow to provide context is key to achieving high accuracy in the queries generated by language models. Context refers to the information we provide to the model as an aid to understanding the data we are working on.

We can distinguish between three types of contextual data: the schema itself (adding only the schema DDL, i.e. CREATE TABLE statements), static examples (adding sample SQL queries from our database to the schema), and contextually relevant examples (adding only the most relevant information, such as SQL queries, DDL statements, or any database-related documents extracted using embedding-based vector search).

Enterprise data warehouses often contain hundreds or even thousands of tables and orders of magnitude more queries covering all the use cases in their organizations. Given the limited size of language model context windows, we are unable to use all previous queries and schema definitions in the system prompt.

In the approach used by Vanna.AI, only relevant data (SQL queries, tables, documents) are selected based on the user's query, which allows for efficient context management and increases the precision of answers.

We can write three types of metadata to the vector database for training the RAG layer and providing better context in the prompt: DDL queries, SQL queries and documentation. We can enter all this data into the model before the application is initialized, or using a built-in tool already in the application.

The most basic option is to provide the database schema in the form of DDL, which will be used to prepare the structure of the database, and through it the model can get acquainted with the fields and field types of each table. A special case of DDL queries are documents generated from the database schema (e.g., the result of a SELECT * FROM INFORMATION_SCHEMA.COLUMNS query), which can be used as a training plan for the model.

The second option is to give the model SQL queries relating to the data we have. When you enter an SQL query, it is added together with a question generated by the model based on it. If the query generated in this way is indeed correct, it can be used as training data in conjunction with the query given earlier. In addition, while working with the chat, if the user thinks that the assistant-generated answer with SQL is correct, it is possible to enter the query-SQL pair as training data for further refinement.

The last type of training data is the so-called documentation. These are all kinds of descriptions that can provide useful context for our data, such as general information about what data is in our database, characteristics of individual tables and fields, or explanations of specific industry terms. With the context provided, the model will be able to return more precise answers to the user, better interpreting the questions asked.

The implementation of Vanna.AI is very simple thanks to the functions and classes prepared in it. Launching a fully operational tool requires only a few steps. First, we define an object of the class, in which we specify the model and vector base used, and initialize it with the appropriate parameters, such as the address where Ollama is exposed (if it is not running on the local machine), the name of the model and possibly its additional parameters, such as temperature.

```
class MyVanna(ChromaDB_VectorStore, Ollama):
    def __init__(self, config_ollama=None, config = None):
        ChromaDB_VectorStore.__init__(self, config=config)
        Ollama.__init__(self, config=config_ollama)

vn = MyVanna(config_ollama={'ollama_host':"http://10.10.10.52:11434",
                            'model': 'gemma2:27b',
                            'ollama_options': {"temperature":0}})
```

Then we connect to the target database using the appropriate connection string.

```
conn_str = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={database["server"]};DATABASE={database["database"]};UID={database["user"]};PWD={database["password"]}'
vn.connect_to_mssql(odbc_conn_str=conn_str)
```

After following these steps, we already have the object connected to all the key elements: the language model, the vector database and the database to which we will ask questions.

At this stage, we can proceed to train the RAG layer. First, we download the database schema and create a training plan based on it, which we then use for training.

```
df_information_schema = vn.run_sql("SELECT * FROM INFORMATION_SCHEMA.COLUMNS")
plan = vn.get_training_plan_generic(df_information_schema)

vn.train(plan=plan)
```

Now we can run an application (e.g. using Flask) with the text2sql functional assistant. If we do not want to use the default applications prepared by the Vanna.AI team, we can use the functionality to generate responses or code directly by calling the appropriate methods.

```
from vanna.flask import VannaFlaskApp
app = VannaFlaskApp(vn, allow_llm_to_see_data=True)
app.run()
```

or

```
vn.ask(question="What are sales in june 2018?")
vn.generate_sql(question="What are sales in june 2018?")
```

The described solution uses the following technologies: a Gemma 2 language model exposed on Ollama, a ChromaDB vector database with default embedding and a Microsoft SQL Server database. This combination makes it possible to efficiently generate SQL queries based on natural language queries tailored to the specifics of the database being used.

Vanna.AI provides a powerful tool for developers and data analysts, enabling fast and accurate generation of SQL queries using modern language models. With the ability to customize the context and continuously refine the model, the tool can significantly improve the work with databases, especially in environments with a complex structure and a large number of tables.

3. Methodology

The purpose of this study was to compare the effectiveness of three selected large language models (LLMs): Llama3:70b-instruct, Gemma2:27b and Codegemma, in generating correct SQL queries based on questions formulated in natural language. The analysis aimed to identify the optimal solution for further collaboration with Vanna.AI, focusing on the models' ability to interpret and translate user queries into precise SQL commands.

All models received identical training data to ensure a fair and consistent comparison. The data consisted of five question-SQL pairs that included a variety of query types, such as aggregate functions, subqueries and filter conditions. This was to evaluate the models' ability to handle different query structures and complexities.

In addition, the models were provided with training information to help them generate SQL queries correctly. This information included details on date format, where it was determined that dates are stored in "YYYY-MM-DD" format, and the models were instructed not to convert dates from "varchar" to "date" type. In addition, models were instructed to use the condition "WHERE Knt_KodP LIKE '[0-9][0-9]-[0-9][0-9][0-9]'" when querying Polish customers.

A detailed database structure was also provided, which included two tables: "ERP_Claims_Orders," storing information about claims to orders, and 'Orders1_new_columns,' containing information about orders. DDL commands for these tables were also included to provide a complete structural context for the database.

The training data prepared in this way was intended to provide a solid basis for evaluating the models' ability to generate correct and effective SQL queries based on natural language queries, taking into account the specific requirements and database structures used in practice.

Six benchmark questions were used to evaluate the models:

1. On which day in October 2018 were the most complaints filed? (A question similar to one from the training data).
2. Give all the details of these complaints that are related to orders processed in the city of Lublin.
3. What percentage of total revenue were orders placed by Polish customers?
4. What percentage of the total revenue were orders placed by customers outside Poland?
5. For which contractors the total value of orders in 2019 was greater than or equal to 100 thousand?
6. What total revenue was generated by orders placed in the last quarter of 2020?

The study was conducted in two iterations to assess the impact of additional database schema information on the performance of the models.

Iteration One (Figure 2):

- Training: The models were trained using the initial training data and additional information provided in the instructions.
- Testing: Models were given benchmark questions to generate appropriate SQL queries.
- Evaluation: The correctness of the generated SQL queries was evaluated against the expected results.
- Time Measurement: For each question, the model's response generation time was measured by taking two measurements.

Iteration Two:

- Enhanced Training Data: The training data was enhanced with a database schema generated using the get_training_plan_generic() function, using the result of the SELECT * FROM INFORMATION_SCHEMA.COLUMNS query.
- Testing and Evaluation: The same benchmark questions were used again, and the models' responses were evaluated in the same way as in the first iteration.
- Time Measurement: Again, the time for models to generate responses was measured.

Evaluation Metrics:

- Correctness: Evaluation of the correctness of the generated SQL queries in the context of the questions asked.
- Response Time: Measurement of the time it takes for models to generate responses, expressed in seconds.

4. Research Results and Discussion

The results of the first iteration showed that the Llama3 and Gemma2 models correctly answered 5 out of 6 questions, while Codegemma correctly answered 4 out of 6 questions. All models encountered the same problem: they were unable to correctly answer question number 5 about contractors with a total order value in 2019 greater than or equal to 100 thousand zlotys. As for the time to generate answers, the models showed mixed results.

For the first question, Llama3 took 10.8 and 5.6 seconds respectively in two measurements, Gemma2 took 10.7 and 2.6 seconds, and Codegemma took 5.1 and 1.4 seconds. For the second question, Llama3 consumed 9.4 and 4.4 seconds, Gemma2 9.2 and 2.3 seconds, and Codegemma 2.1 and 1.4 seconds. Similar differences were observed for the following questions, where response generation times ranged from a few to several seconds, with Codegemma typically showing the shortest response times.

Response generation times for the two measurements per question (in seconds) are shown in Table 1.

Table 1. *Response generation times per question for two measurements*

Question No.	Llama3	Gemma2	Codegemma
1	10,8 / 5,6	10,7 / 2,6	5,1 / 1,4
2	9,4 / 4,4	9,2 / 2,3	2,1 / 1,4
3	12,6 / 12,4	6,5 / 6,4	2,9 / 2,8
4	13,4 / 13,2	6,2 / 6,1	3,0 / 2,9
5	11,4 / 8,4	4,8 / 4,9	2,1 / 1,7
6	6,9 / 6,7	2,4 / 2,4	1,3 / 1,4

Source: *Own study.*

The results of the second iteration showed that the Llama3 model correctly answered only one of the six questions, the same one it had difficulty with in the previous iteration. The Gemma2 model answered two of the six questions correctly. Codegemma also answered only one of the six questions correctly. An analysis of the answer generation times in this iteration is as follows:

For question one, the Llama3 model took 28.3 and 13.3 seconds in two measurements, Gemma2 took 74.7 and 2.1 seconds, respectively, and Codegemma took 4.7 and 1.5 seconds. For question two, Llama3 consumed 21.7 and 13.3 seconds, Gemma2 3.5 and 2.4 seconds, and Codegemma 3.9 and 1.0 seconds. For the third question, the times were: Llama3 16.1 and 12.7 seconds, Gemma2 3.2 and 2.4 seconds, and Codegemma 2.1 and 2.5 seconds.

Continuing, for the fourth question, Llama3 took 16.1 and 16.0 seconds, Gemma2 4.6 and 3.5 seconds, and Codegemma 2.6 and 2.2 seconds. For the fifth question, the times were: Llama3 11.9 and 9.1 seconds, Gemma2 3.7 and 2.6 seconds, and Codegemma 1.9 and 1.5 seconds. Finally, for the sixth question, Llama3 used 11.6 and 28.2 seconds, Gemma2 used 3.3 and 2.5 seconds, and Codegemma used 1.5 and 1.2 seconds.

These results indicate that adding a detailed database schema in the second iteration did not improve the performance of the models, but rather the opposite – Table 2. The models may have been overwhelmed by excess information, which negatively affected their ability to generate correct answers and their response times.

Table 2. *Response generation times per question for two measurements – second generation*

Question No.	Llama3	Gemma2	Codegemma
1	28,3 / 13,3	74,7 / 2,1	4,7 / 1,5
2	21,7 / 13,3	3,5 / 2,4	3,9 / 1,0
3	16,1 / 12,7	3,2 / 2,4	2,1 / 2,5
4	16,1 / 16,0	4,6 / 3,5	2,6 / 2,2

5	11,9 / 9,1	3,7 / 2,6	1,9 / 1,5
6	11,6 / 28,2	3,3 / 2,5	1,5 / 1,2

Source: *Own study.*

Overall observations from the survey indicate that adding detailed information about the database schema did not provide the expected benefits. Models had difficulty understanding equivalent terms such as “contractor” and “customer,” suggesting the need for better training in interpreting domain terminology. In addition, an increase in response generation time was observed, particularly for the Llama3 model, suggesting that models need to be optimized for performance.

The study found that providing LLM models with overly detailed database schema information does not necessarily translate into better performance in generating valid SQL queries. In some cases, this can lead to overloading of the models and deterioration of the quality of the generated answers. As a result, models may require further tuning and better training in interpreting database schema information.

Recommendations for the future include considering providing simplified database schema information, which can make it easier for models to efficiently process key data without overloading. It is also important to include explicit mappings of synonymous terms to training data so that models can better understand the specific terminology used in the domain. Continued work on tuning models to task specificity is key to improving their performance and accuracy.

In summary, choosing the optimal LLM model for generating SQL queries requires finding a balance between accuracy, response time and the model's ability to process additional information. This study underscores the complexity of this task and the need for careful selection of training data and an in-depth understanding of the models' capabilities to achieve optimal results.

5. Conclusions

This paper focuses on comparing three LLM models in the context of query processing based on an example involving customer service employees. The main purpose of the paper was to investigate how different models handle accessing key data by asking natural language questions without the need for SQL knowledge.

The study evaluated three different LLM models - Llama3:70b-instruct, Gemma2:27b and Codegemma, as well as the Vanna.AI tool of the open sorce solution in which LLM models can be implemented. The purpose of the study was primarily to identify the benefits and limitations of their advanced natural language processing capabilities for generating SQL queries. Particular attention was paid to the impact of providing detailed information about the database schema on the accuracy of generated queries and the response time of the models.

The research methodology was based on a comprehensive analysis in which models were tested on six benchmark questions. The tests were conducted in two iterations to observe the effect of additional schema data on query generation. All models were trained on identical datasets to generate valid SQL queries reflecting different levels of complexity, from basic queries to nested structures and aggregate functions. Performance was measured using two key metrics: the correctness of the generated SQL and the response generation time.

The results showed that while Llama3 and Gemma2 initially achieved high accuracy, adding detailed information about the database schema did not improve their performance, and in fact contributed to increased response times and decreased accuracy for certain types of queries. It has been observed that excessive context can overload models rather than provide useful improvements.

The Codegemma model, while having slightly lower accuracy than the other models, maintained shorter response times in both iterations. The results suggest that Codegemma may be a practical solution where speed over accuracy is a priority. Overall, the study indicates that it is crucial to provide LLM models with the optimal amount of context to maximize their performance.

In terms of practical implications, the article suggests that customer support should consider providing simplified database schema information to prevent overloading of models while enabling accurate SQL query generation. Implementing tools such as Vanna.AI, using Retrieval Augmented Generation (RAG), can play a key role in optimizing model performance by selectively providing relevant contextual data to avoid information overload problems.

Future research should focus on tuning models for complex query types and developing robust training processes that enable LLM models to adapt to specific industry terminologies. Further work may also explore approaches that improve models' understanding of database schemas without overloading them.

Based on the results presented here, future research could incorporate additional LLM models and RAG techniques to broaden the scope of model performance evaluation, especially across domains and database structures. As Text2SQL technology evolves, targeted optimizations and iterative learning will be key to fully realize the potential of LLM for dynamic customer service needs, enabling faster and more accurate access to data across a wide range of applications.

References:

Beurer-Kellner, L., Fischer, M., Vechev, M. 2023. Prompting Is Programming: A Query Language for Large Language Models. Proceedings of the ACM on Programming Languages, 7(PLDI), 1946-1969. https://doi.org/10.1145/3591300.

Biswal, A., Patel, L., Jha, S., Kamsetty, A., Liu, S., Gonzalez, J. E., Guestrin, C., Zaharia, M. (n.d.). Text2SQL is Not Enough: Unifying AI and Databases with TAG. In: Proceedings of ACM Conference (Conference'17) (Vol. 1). https://github.com/TAG-Research/TAG-Bench.

Davis, F., Easton, H. 2015. Financial Crisis and Banking Performance. The International Journal of Finance, 12(2), 34-48.

Dima, A., Lukens, S., Hodkiewicz, M., Sexton, T., Brundage, M.P. 2021. Adapting natural language processing for technical text. Applied AI Letters, 2(3). https://doi.org/10.1002/ail2.33.

Fu, H., Liu, C., Wu, B., Li, F., Tan, J., Sun, J. 2023. CatSQL:Towards Real World Natural Language to SQL Applications. Proceedings of the VLDB Endowment, 16(6), 1534-1547. https://doi.org/10.14778/3583140.3583165.

Li, C., Wang, Y., Wu, Z., Yu, Z., Zhao, F., Huang, S., Dai, X. 2024. MultiSQL: A Schema-Integrated Context-Dependent Text2SQL Dataset with Diverse SQL Operations. Findings of the Association for Computational Linguistics ACL 2024, 13857-13867. https://doi.org/10.18653/v1/2024.findings-acl.823.

Lin, W.Y. 2023. Prototyping a Chatbot for Site Managers Using Building Information Modeling (BIM) and Natural Language Understanding (NLU) Techniques. Sensors, 23(6), 2942. https://doi.org/10.3390/s23062942.

Liu, C., Zhang, W., Zhao, Y., Luu, A.T., Bing, L. (n.d.). Is Translation All You Need? A Study on Solving Multilingual Tasks with Large Language Models. https://platform.openai.com/docs/models/.

Mahmood, A., Yao, B., Huang, C.M. 2024. LLM-Powered Conversational Voice Assistants: Interaction Patterns, Opportunities, Challenges, and Design Guidelines.

Praveen, S.V., Gajjar, P., Ray, R.K., Dutt, A. 2024. Crafting clarity: Leveraging large language models to decode consumer reviews. Journal of Retailing and Consumer Services, 81, 103975. https://doi.org/10.1016/j.jretconser.2024.103975.

Song, Y., Ezzini, S., Tang, X., Lothritz, C., Klein, J., Bissyande, T., Boytsov, A., Ble, U., Goujon, A. 2024. Enhancing Text-to-SQL Translation for Financial System Design. Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, 252-262. https://doi.org/10.1145/3639477.3639732.

Tomova, M., Hofmann, M., Hütterer, C., Mäder, P. 2024. Assessing the utility of text-to-SQL approaches for satisfying software developer information needs. Empirical Software Engineering, 29(1). https://doi.org/10.1007/s10664-023-10374-z.

Yan, B., Li, K., Xu, M., Dong, Y., Zhang, Y., Ren, Z., Cheng, X. 2024. On Protecting the Data Privacy of Large Language Models (LLMs): A Survey. http://arxiv.org/abs/2403.05156.

Yang, E., Nair, S., Lawrie, D., Mayfield, J., Oard, D.W., Duh, K. 2024. Efficiency-Effectiveness Tradeoff of Probabilistic Structured Queries for Cross-Language Information Retrieval. http://arxiv.org/abs/2404.18797.

Zhang, L., Jijo, K., Setty, S., Chung, E., Javid, F., Vidra, N., Clifford, T. 2024. Enhancing Large Language Model Performance To Answer Questions and Extract Information More Accurately.

Zulfikar, W., Chan, S., Maes, P. 2024. Memoro: Using Large Language Models to Realize a Concise Interface for Real-Time Memory Augmentation. Conference on Human Factors in Computing Systems - Proceedings. https://doi.org/10.1145/3613904.3642450.