SQLAssist: A RAG unified framework for Natural Langauge to SQL translation

BASAVARAJ SANGAYYA HIREMATH, DHANASHREE GUNDA*, MALLIKA MANAM*, NIHARIKA RAMANATH*, and SUPRIYA P UPADHYAYA*, Otto-von-Guericke-Universität Magdeburg, Germany

Proficiency in the Structured Query Language(SQL) is often necessary when interacting with the database effectively which is often difficult for the non-expert users. Text-to-SQL aims to address this issue by automating SQL query generation from natural language text, leveraging neural network architectures and innovative NLP techniques. There is interest in using large language models (LLMs) for code interpretation because of their impressive performance in this area, as shown by BERT and GPT4. Nevertheless, there are difficulties in implementing LLMs, including large computational and storage needs. In order to overcome these obstacles, our framework utilizes a unified multi-module framework that integrates the Parameter-efficient Fine-Tuning (PEFT) method into the llama-3-8b 4bit quantize model, which allows for effective fine-tuning with a small amount of computational power. The main component of our SQLAssist framework is its unified approach that leverages a RAG based to gather relevant contextual information such as indexed tables and sample rows. This enriched context empowers the model to generate accurate SQL queries. To validate the generated queries, we integrate a Refiner tool that detects and rectifies SQL errors. By making these improvements, our methodology raises the efficiency of text-to-SQL translation and advances the usability of databases for non-expert users. Experiments show that our approach achieves a comparable BLEU and ROGUE score of 0.384 and 0.729 respectively when evaluated on the open-source hoangphu7122002ai/text2sql_en dataset. We also observed that the approach works well for basic sql complexity.

Additional Key Words and Phrases: Large Language Models (LLM), LLama, Text-to-SQL, Natural language Processing (NLP)

1 INTRODUCTION

Effective communication with the database requires knowledge of Structured Query Language (SQL), which presents a significant challenge to non-expert users[4]. Text-to-SQL attempts to make databases more accessible by automating the creation of SQL queries from normal language text, all without requiring the user to be an expert in SQL query formulation[11]. State-of-the-art methods in Text-to-SQL have often involved a combination of neural network architectures with innovative techniques in natural language processing(NLP) and database query understanding. Large language models (LLMs) such as BERT and GPT4 have shown remarkable performance in various NLP tasks such as generating the SQL queries from the natural language input[1] which has further sparked the interest in leveraging LLMs for code understanding and generation.

Our research aims to overcome the challenges faced while deploying LLMs particularly the need for extensive storage and computational resources. The PEFT (Parameter-efficient fine-tuning) technique[6] is incorporated into our llama-3-8b-bnb-4bit-text-to-sql model to allow for efficient fine tuning in a short amount of time with a minimum amount of computational resources. At the core of our paper is the Fine tuning of the LLama3 using a method called Lora (Large-scale Language Model Low-Rank Adaptation)[8]. LoRA enables the adaptation of pretrained language models to specific tasks or domains by augmenting them with an auxiliary branch that learns task-specific information. LoRA strikes a compromise between task efficacy and computational efficiency by training only a subset of parameters while maintaining the essential knowledge embodied in the pretrained model. This approach allows for more efficient fine-tuning of large language models, making them more suitable for a wide range of downstream applications[9].

The Unsloth library is a state-of-the-art tool that we used to speed up training considerably, and help us fine-tune language models. All of these improvements increased the effectiveness of our model's learning process. To verify the queries that are generated by the model we have integrated a Refiner[14] whose primary function is to detect and rectify the SQL errors.

To streamline the text-to-SQL process and mitigate the impact of extraneous data, we incorporate the Retriever module[5], which automatically splits the database into smaller, easier-to-manage sub-databases, especially in cases involving huge databases with lots of tables and columns. This segmentation procedure is essential for increasing productivity and minimising noise from unrelated data. There are two main components to the Retriever module: Retrieve Tables in Query Time and Retrieve Sample Rows in Query Time. Having laid the foundation with an overview of the text-to-SQL task and our research objectives, we now proceed to the evaluation section, where we rigorously assess the performance of our llama-3-8b-bnb-4bit-text-to-SQL model. Following the evaluation, we delve into a detailed analysis of the results and discuss their implications in the subsequent sections.

2 PRIOR RELATED WORK

2.1 LLM-Based Text-to-SQL

Recent advancements in large language model (LLM)-based text-to-SQL tasks have focused on refining prompt designs and developing multi-stage frameworks for improved performance. Initially, the emphasis in LLM research was on crafting high-quality prompts to better utilize LLMs for SQL generation. For instance, Bing Wang et al[14]. introduced a framework featuring a core decomposer agent employing few-shot chain-of-thought reasoning to generate SQL queries from text, complemented by auxiliary agents utilizing tools or models to fetch subdatabases and rectify incorrect SQL queries. Similarly, Tai et al[13] explored chain-of-thought type prompting strategies like least-to-most and original chain-of-thought prompting to enhance LLMs' reasoning capabilities.

Recent work on LLM-based Text-toSQL has mostly focused on supervised fine-tuning with data from the target domain and In-Context Learning prompt techniques. However, with "huge" datasets and complicated user inquiries requiring multi-step reasoning, these techniques typically suffer from severe performance deterioration. Furthermore, the majority of current approaches ignore how important it is for LLMs to collaborate with models and use external tools.

Studies such as DIN-SQL and StructGPT have proposed frameworks incorporating zero-shot approaches, query decomposition techniques, and specialized interfaces for structured data access. These frameworks aim to simplify databases, generate SQL queries, verify query correctness, and integrate answers. The multi-agent collaboration architecture proposed by Bing Wang et al[14] offers versatility in handling complex data scenarios and identifying a wider range of errors for correction.

Present STAR[2], a pre-trained Table-aware Language Model (TaLM) designed for context-dependent text-to-SQL dialogues, in their most recent work. Two novel pre-training objectives, utterance dependency tracking and schema state tracking, are built into STAR to capture the intricate relationships that occur between natural language (NL) utterances and SQL queries throughout a conversation. By utilising a broad and well-curated large-scale corpus for context-dependent text-to-SQL dialogues, STAR attains significant improvements in performance, setting new benchmarks on COSQL and SPARC datasets. The effectiveness of STAR in modelling contextual nuances and its significant improvement in performance across difficult text-to-SQL tasks demonstrate the importance of this work in furthering text-to-SQL conversational systems.

2.2 Fine-tuning methods for LLMs

The two primary categories of tuning techniques are reinforcement learning and supervised fine-tuning[10]. In order to improve performance, supervised fine-tuning trains models on certain task datasets that are suitable for Natural Language Understanding (NLU)[7]. The appropriate tuning strategies and high-quality data play a pivotal role in the performance of LLMs[12].

Liu et al[9] identify the key challenges in NLP fine-tuning and propose MFTCoder (multi-task fine-tuning framework) to enhance efficiency and performance. This study focuses on the Parameter-Efficient Fine-Tuning

(PEFT) which optimizes the model parameters for maximal impact and minimal computational overhead. There are two PEFT methods namely: LoRA(Large-scale Language Model Low-Rank Adaptation)[8] and QLora (Quantized Large-scale Language Model Low-Rank Adaptation)[3]. Building on LoRA, QLoRA quantizes the pretrained model to 4 bits using dual quantization and a revolutionary high-precision quantization method called NF4.

3 METHODOLOGY

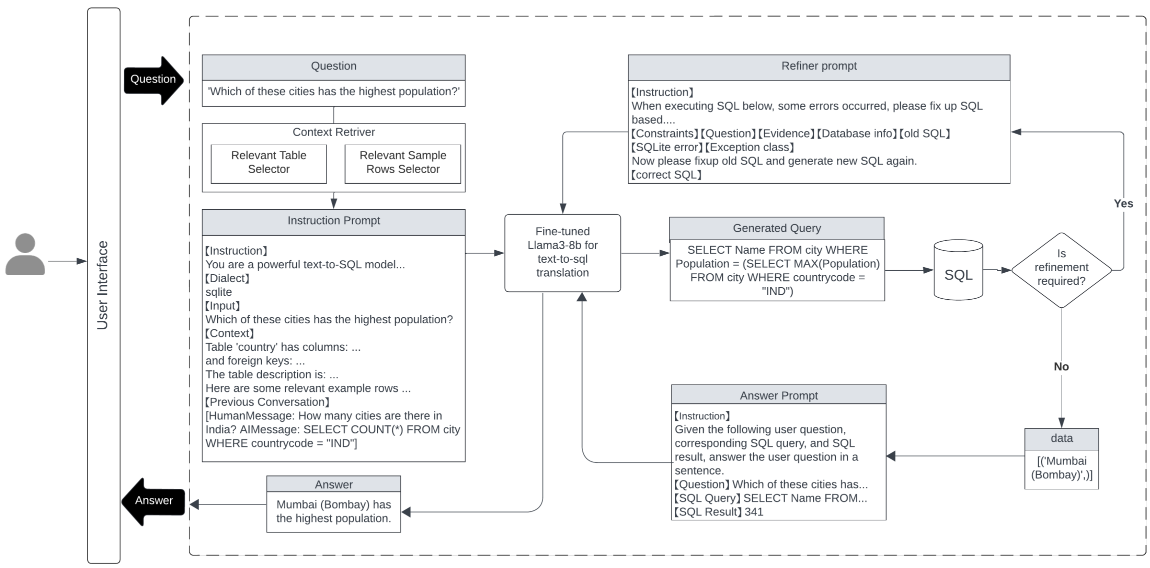

Fig. 1. The overview of our SQLAssist framework

3.1 Datasets

Our selection was made using the 262,208-item Clinton/Text-to-sql-v1 dataset available on Huggingface. It is composed of multiple datasets like wikiSQL, sql-create-context, nvbench, mimicsql-data, squall, mimic-iii, sede, eicu, spider, atis and advising.

In order to evaluate the overall performance of our model, we use two datasets from Hugging Face: hoangphu7122002ai/text2sql-en and gretelai/synthetic-text-to-SQL. The latter dataset offers a wide variety of queries without particular complexity labels, whereas the former collection is categorised based on SQL query complexities. We evaluate 1000 items from each dataset to ensure a robust analysis of our model's capabilities. The gretelai/synthetic-text-to-SQL dataset includes queries categorized into various complexity levels, such as 'basic SQL' and advanced categories like aggregation, Common Table Expressions (CTEs), Multiple Joins, Set Operations, Subqueries, and Window Functions. The hoangphu7122002ai/text2sql-en dataset, on the other hand, provides a wider range of queries without predetermined complications. By evaluating both datasets, we aim to provide a comprehensive assessment of our model's performance across different query scenarios and complexities.

3.2 Overview

Our proposed framework for a robust text-to-sql task is described in the figure 1 above. The user interacts with the streamlit web application that we developed to query any database that it is connected to. When the user prompts a question, the query pipeline is initiated. The first module in the pipeline is the context retriever which retrieves the tables relevant to the user query and also fetches some example rows that might aid in the proper sql generation. These tables and rows are then merged to form the context and the instruction prompt is prepared which contains the instruction for the finetuned model. The prompt also includes the history of the conversation as depicted in the figure above. The model generates the SQL query when the instruction prompt is fed in.

Furthermore, the SQL database agent tries to execute the query to produce a response. If the executor throws any error while trying to execute the generated query, it routes it to the refiner module. If there the query is successfully executed, the rephrase module takes this response and the user query to generate a response back to the user in the natural language. When the refiner module is invoked, it prompts the finetuned model again with the refiner prompt which tries to regenerate the query with 5 maximum retries until it generates a valid sql query. If the retries are execeed, then it returns a sorry message to the user and requests to rephrase the prompt. When the retriever generates a valid query, the pipeline enters the rephrase module and the flow continues to generate the response to the user.

In the following section, a detailed introduction of three main modules introduced in our approach will be presented.

3.2.1 ***Context Retriever.*** In order to reduce interference from unnecessary information, the Retriever module is built to automatically split up large databases into smaller subdatabases. In order to reduce influence from unnecessary data, the Retriever is built to do an initial filtering of pertinent tables and columns. There are two parts to the Context Retriever:

(1) Query-Time Table Retrieval : If we are unsure about which table to use in advance and the overall size of the table schema exceeds the context window size, we store the table schema in an index to retrieve the correct schema during query time.
(2) Query-Time Sample Row retrieval: There would be no response to an input query with a value that didn't match with what was stored in the database. This issue may be resolved by retrieving a limited set of sample rows for each table. Taking the first k rows would be a simple option. Instead, in order to provide the text-to-SQL LLM with the most contextually relevant information for the user query, we embed, index all the rows of all the tables, and extract only k relevant rows.

3.2.2 ***Refiner.*** In the context of Text-To-SQL tasks, the main function of the Refiner is to detect and fix the erroneous SQL queries. As shown in Figure 1, on receiving the SQL query the Refiner initiates an evaluation process. Firstly, it checks for syntactical correctness and verifies the retrieval of non-empty results from the database. If the query passes these checks, it proceeds to convert the SQL query into natural language format, serving as the final answer output. In cases where the query is found to be incorrect, the Refiner undertakes correction operations to rectify the errors. This corrected query then undergoes re-evaluation, with the process iterated up to five times or until the expected result is attained. Notably, the Refiner contributes significantly to the reduction of syntax errors and other query-related issues. The other issues are often misunderstandings in database handling, misinterpretation of evidence and understanding of the questions which pose greater challenges.

3.2.3 ***Rephraser.*** Presenting the data in a way that is easy for users to grasp is crucial, when NL2SQL model answers a SQL query correctly. This is where the skill of translating SQL findings into understandable, plain language responses becomes useful. To help the model rephrase SQL findings, we have developed a prompt template. These templates may be used to set up the process of creating a natural language answer by include placeholders for the original question, the SQL query, and the query result.

3.3 Instruction-tuned Llama-3-8b

We have open-sourced Llama-3-8b and fine-tuned our model utilizing Clinton/Text-to-sql-v1 data, improving its skills in SQL creation, SQL correction, and database simplicity. We faced many significant difficulties when training the model, one of which was striking a compromise between model complexity and performance. To guarantee that the model could manage the intricacies of database-related activities while retaining high performance levels, we had to carefully tune the model architecture and parameters.

3.4 Memory Integration

Adding memory to the chatbot is one of the most sophisticated ways to create an NL2SQL interface that is easy for users to navigate. With the help of this function, the chatbot intelligently responds to follow-up queries about the database, giving users a smooth conversational experience. Context is important in real-world scenarios. It might be possible that a question is related to previous interaction. In a similar fashion, users' subsequent inquiries while interacting with a database via chatbot frequently rely on the context information provided by the previous queries and answers. This context is remembered by a chatbot that has memory, which enables it to provide more precise and pertinent SQL queries for follow-up inquiries. The message history is included in prompt generation process and the model considers the history to understand the context of the conversation which guides the model in understanding the follow-up questions.

3.5 Implementation Details

We leveraged the Google Colab platform which provides free Tesla T4 GPU with 15GB RAM. The training and inference was performed using the FastLangaugeModel module from unsloth library which accelerates the training and inference by 2 fold. We finetuned the Llama-3-8b 4 bit quantized optimized for finetuning provided by unsloth. The training data used a combination of spider, wikisql and a few more datasets which had 262,208 samples. The training was done for only 100 steps with learning rate of 2e-4 and adamw-8bit optimizer with a batch size of 8. The model was finetuned for q, k,v,0,gate,up and down projection layers. The total training parameters were 41,943,040. The training and validation loss for 100 steps of training were around 0.4 for both. We also experimented with various other models like tinyLlama, Llama-2 and Mistral-7b with the same configuration as above.

4 EXPERIMENTS

4.1 Experimental Setup

Evaluation Metrics: Utilizing the gretelai/synthetic-text-to-SQL dataset, our evaluation of the llama-3-8b-bnb-4bit-text-to-SQL model emphasizes the importance of two key evaluation metrics: BLEU score and ROUGE score. The Bleu score serves as a measure of how closely the generated SQL query aligns with the ground truth provided in the dataset. A score of 1 signifies a perfect match between the generated query and one of the reference queries, offering a numerical output ranging from 0 to 1.

The ROUGE score provides the quantitative assessment of the model by comparing the quality of the model-generated SQL queries to the dataset's ground truth queries. Higher ROUGE scores indicate a closer alignment between the model outputs and the expected queries, making it a useful indicator to assess the relevance and correctness of the text-to-SQL conversion process. By incorporating the ROUGE score into our evaluation framework, we gain valuable insights into the performance of our model, facilitating informed decisions for further refinement and optimization.

User Question: How many cities in the USA? Generated SQL Query: SELECT COUNT(*) FROM city WHERE countrycode = "USA" SQL Result: [(274,)] Answer: There are 274 cities in the USA. User Question: How many cities in the USA? Previous conversation :
User Question: Which of these cities has the highest population? Generated SQL Query: SELECT Name FROM city WHERE Population = (SELECT MAX(Population) FROM city) SQL Result: [('Mumbai (Bombay)',)] Answer: Mumbai (Bombay) has the highest population. User Question: Which of these cities has the highest population? Previous conversation : [HumanMessage(content='How many cities in the USA?'), AIMessage(content='SELECT COUNT(*) FROM city WHERE countrycode = "USA"')]
User Question: Which of these cities has the lowest population? Generated SQL Query: SELECT Name FROM city WHERE Population = (SELECT MIN(Population) FROM city) SQL Result: [('Adamstown',)] Answer: Adamstown has the lowest population. User Question: Which of these cities has the lowest population? Previous conversation : [HumanMessage(content='How many cities in the USA?'), AIMessage(content='SELECT COUNT(*) FROM city WHERE countrycode = "USA"'), HumanMessage(content='Which of these cities has the highest population?'), AIMessage(content='SELECT Name FROM city WHERE Population = (SELECT MAX(Population) FROM city)')]
User Question: What are the unique districts of these cities? Generated SQL Query: SELECT DISTINCT District FROM city WHERE Name IN (SELECT Name FROM city WHERE CountryCode = "USA") SQL Result: [('Paraná',), ('England',), ('Bíobío',), ('Aleksandria',), ('Santa Ana',)] Answer: The unique districts of these cities are Paraná, England, BíoBío, Aleksandria, and Santa Ana. User Question: What are the unique districts of these cities? Previous conversation : [HumanMessage(content='How many cities in the USA?'), AIMessage(content='SELECT COUNT(*) FROM city WHERE countrycode = "USA"'), HumanMessage(content='Which of these cities has the highest population?'), AIMessage(content='SELECT Name FROM city WHERE Population = (SELECT MAX(Population) FROM city)'), HumanMessage(content='Which of these cities has the lowest population?'), AIMessage(content='SELECT Name FROM city WHERE Population = (SELECT MIN(Population) FROM city)')]

Fig. 2. Follow-up query

In essence, our evaluation seeks to quantify the accuracy and fidelity of our model's output by assessing its resemblance to the ground truth queries present in the dataset. Through this rigorous evaluation procedure, we are able to determine how well our llama-3-8b-bnb-4bit-text-to-SQL model performs in producing SQL queries that closely resemble reference queries created by humans.

4.2 Qualitative Analysis

We performed the following experiments on our finetuned Llama-3-8b finetuned model.

(1) Follow-up query: In this experiment, we explored the ability to answer follow-up questions based on the generated SQL query. This is an important aspect of building a truly interactive and helpful system, as users might have further questions after seeing the initial query. We observed that sometimes if fails to efficiently make use of the history to identify the root context to answer the current question as shown in the image below.

(2) Query Complexity (Fig 3): To evaluate model's ability to generate complex queries, we framed complex questions that will include including joins and aggregation functions. The model was able to generate query that will fetch correct answers but using sub-queries instead of join functions. We suspect this could be because of the insufficient training examples with joins.

Question : Which is the most spoken language of the largest country in Asia by surface area?
Generated SQL Query : SELECT Language FROM countrylanguage WHERE CountryCode = (SELECT Code FROM country WHERE Continent = "Asia" ORDER BY SurfaceArea DESC LIMIT 1) ORDER BY Percentage DESC LIMIT 1 **SQL Result :** [('Chinese',)] **Answer :** Chinese is the most spoken language of the largest country in Asia by surface area. **Previous conversation :** [HumanMessage(content='Which is the largest country in Asia by surface area?'), AIMessage(content='SELECT Name FROM country WHERE Continent = "Asia" ORDER BY SurfaceArea DESC LIMIT 1'), HumanMessage(content='Which is the most populated city in the Karnataka district of India'), AIMessage(content='SELECT Name FROM city WHERE District = "Karnataka" ORDER BY Population DESC LIMIT 1'),] **Is refined :** False **Refined queries :** []

Fig. 3. Query Complexity

(3) Poorly phrased question: To evaluate our fine-tuned text-to-SQL model's robustness against poorly framed questions, we designed experiments focusing on the following common issues:
   - Ambiguous Questions: We created questions containing ambiguous phrases or where the desired aggregation function is unclear (e.g., "Which are the most city?" vs. "List the countries in India")
   - Incomplete Questions: We constructed questions lacking crucial information for generating a complete SQL query (e.g., "Find data about countries").

- Grammatical Errors: We introduced questions with grammatical mistakes or typos (e.g., "List the districts of India").

  The model's performance was measured on its ability to handle these errors and translate the intent into a valid SQL query. During these cases, the framework requests the user to rephrase the query accurately.

(4) Relevant table extraction: we investigate the effectiveness of our fine-tuned text-to-SQL model in identifying the most relevant table for a given query. The model was efficient in retrieving the relevant indexed tables most similar to the based on the input query. By evaluating our model on this task, we gain valuable insights into its ability to understand the relationships between natural language queries and the underlying database schema. This allows us to refine the model for more accurate and efficient data retrieval.

(5) Refinement: To verify the working of refiner module, we experiment with a few queries until the models generated a query which was failed when the SQL database executor tried to generate the response. Below is a snippet which shows how the framework handles the refinement.

Demonstration

User Question: Who is the Head of State of Germany?
Initial Generated SQL Query: SELECT HeadOfState FROM country WHERE Name =
Initial SQL Result: <class'sqlite3.OperationalError'>
Previous conversation: [HumanMessage(content='How many cities are in the USA?'), AIMessage(content='SELECT COUNT(*) FROM city WHERE countrycode = "USA"'), HumanMessage(content='Which is the Head of State of Germany?'), AIMessage(content='SELECT HeadOfState FROM country WHERE Name =')]
Is refined: True
Refined queries: ["SELECT HeadOfState FROM country WHERE Name = 'Germany'"]
SQL Result: [('Johannes Rau',)]
Answer: Johannes Rau is the Head of State of Germany.

Fig. 4. Refined SQL Query

5 RESULTS

Upon analyzing the results for the gretelai/synthetic text-to-sql dataset across various SQL complexity categories, a few important findings have come to light. Firstly, the model demonstrates commendable performance in handling basic SQL queries, as seen by its comparatively high mean BLEU and ROUGE scores. This implies that the model correctly converts natural language queries into SQL commands and successfully captures the essence of simple SQL operations. However, the model's performance tends to change with increasing SQL query complexity. For instance, while the model fares moderately well in handling queries involving Aggregation functions and CTEs, it struggles with queries requiring multiple joins. This suggests that there might be a limit to the model's comprehension and correct representation of intricate join operations.

Remarkably, the model performs better when performing set operations than when addressing single join queries, indicating some level of competency with relational algebraic operations. Notably, the model showcases relatively strong performance in handling queries involving Sub-queries, highlighting its capability to navigate nested query structures. However, its performance in queries involving Window functions is comparatively weaker, indicating a potential area for improvement. Overall, this research offers insightful information about the model's advantages and disadvantages in various SQL complexity categories, suggesting future directions for model optimisation and improvement to increase the model's adaptability and efficiency in managing a range of query circumstances. Despite encountering variability in metrics due to the gretelai/synthetic text-to-sql dataset utilization of aliases in its ground truth references, our model consistently generates equally functional executable queries. The reason for this disparity in metrics is the particular features of the dataset, however when our model runs, the queries it generates return the same answers as the ground truth. This validation underscores

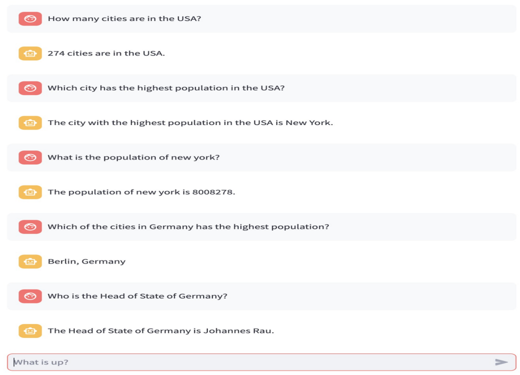

Fig. 5. Model interaction overview

Category	No. Items	Mean BLEU Score	Mean ROUGE Score
Basic SQL	300	0.448117676	0.763702654
Aggregation	100	0.246229027	0.597710224
CTEs	100	0.214314598	0.547419781
Multiple Joins	100	0.061922864	0.448741418
Set Operations	100	0.131161068	0.502536743
Single join	100	0.077280904	0.485317458
Subqueries	100	0.262920038	0.567406656
Window Functions	100	0.074360953	0.468593073

Table 1. Evaluation results for gretelai/synthetic text-to-sql dataset

the effectiveness and reliability of our model in accurately interpreting natural language inputs and producing SQL commands capable of delivering the expected outcomes.

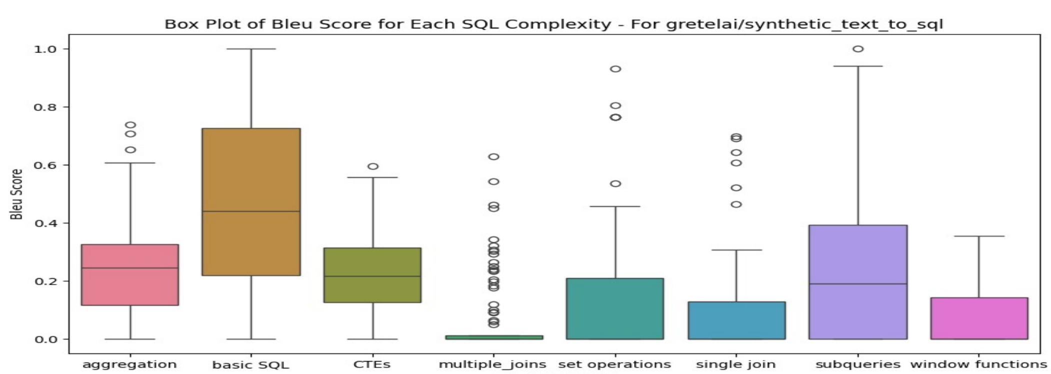

Fig. 6. Box Plot for the Bleu Scores - gretelai/synthetic text-to-sql

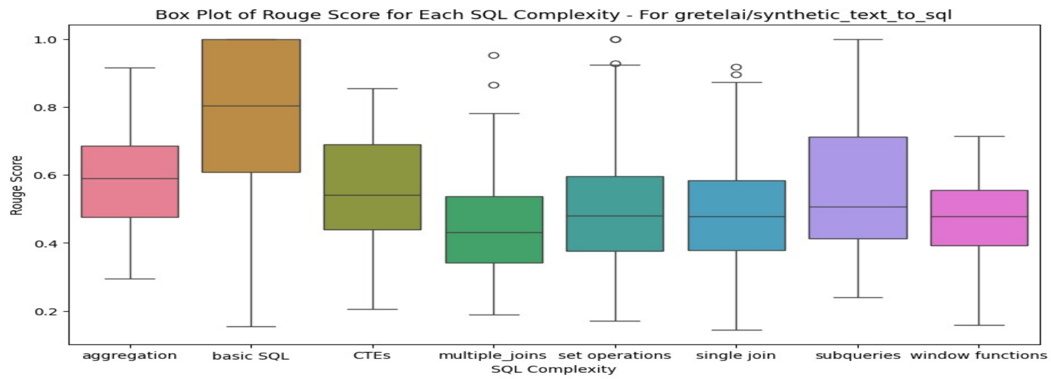

Fig. 7. Box Plot for the Rouge Scores - gretelai/synthetic text-to-sql

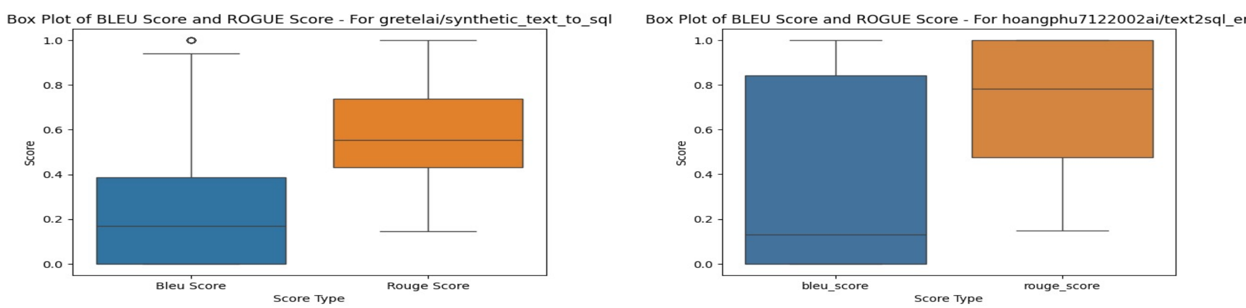

Fig. 8. Box Plot for the Bleu and Rouge Scores

In addition to the comprehensive analysis of the model's performance across various SQL complexity categories, the utilization of box plot (Fig 6 and Fig 7) visualizations provided valuable insights into the distribution and variability of Mean Bleu and Rouge Scores within each category. Box plots allowed for a more nuanced view of the range of model performance, including the presence of outliers and the degree of dispersion within the data, by graphically depicting the distribution of scores.

The **hoangphu7122002ai/text2sql-en** dataset has encouraging performance metrics according to our evaluation results, with mean BLEU and Rouge scores of 0.38469981 and 0.72918639 (Table 2), respectively. The achieved BLEU score indicates a significant alignment between the model-generated queries and the ground truth references, while the Rouge score underscores the quality and relevance of the generated outputs in comparison to the expected queries.

Furthermore, the box plot visualizations facilitated direct comparisons between the distributions of Mean Bleu and Rouge Scores (Fig 8), enabling researchers to identify potential discrepancies or similarities in model performance metrics. The comparative study of box plots, for example, showed situations in which the model performed consistently across the two assessment metrics and instances in which there were significant differences between the Bleu and Rouge Scores within the same SQL complexity category.

Mean BLEU Score	Mean ROUGE Score
0.38469981	0.72918639

Table 2. Evaluation results for hoangphu7122002ai/text2sql-en dataset

6 ERROR ANALYSIS

Infinite/Random Generation: The model sometimes became fixated on memorizing specific training examples, especially with very long contexts or a large number of sample rows and tables. This led to repetitive or nonsensical outputs. Our proposed framework's RAG-based context retriever helps alleviate this problem by selecting relevant information from the input, preventing overfitting to specific training instances.

Difficulty with Complex Queries: The model struggled with complex queries involving joins and aggregations. This likely stems from a lack of training examples with such constructs in the dataset. Expanding the training dataset with a wider variety of complex queries, including joins and aggregations, would improve the model's ability to handle them.

Follow-up Issues: Including all previous messages in the history might overwhelm the model, hindering its ability to identify the most relevant context for the current question. Implementing similarity and attention-based mechanisms can help the model focus on the most crucial information within the conversation history. Additionally, incorporating this approach during training could allow the model to learn how to identify the most relevant or root context for the current query.

Exact Match Metric Limitations: The evaluation metric penalizes queries lacking column aliases, which the model wasn't trained on frequently. Our model likely generates queries without aliases, leading to lower exact match scores. Expanding the training dataset to include more examples with column aliases would improve performance on the exact match metric. Alternatively, the evaluation metric could be adapted to consider semantic equivalence while allowing for minor variations like missing aliases.

Hyperparameter Tuning and Fine-tuning Model Selection: Experimenting with different combinations of LORA target modules could reveal which layer fine-tuning is most effective for text-to-SQL tasks or database tasks in general. Open-source LLMs like tinyLlama, Llama-2, and Mistral-7b proved unsuitable for our scenario, especially when incorporating sample rows and table schemas. Llama-3 emerged as the most effective choice due to its ability to handle complex contexts without infinite generation issues. However, exploring cost-effective ways to utilize GPT models remains an interesting future direction.

7 CONCLUSION

This work proposes the SQLAssist framework, a novel approach to Text-to-SQL translation that leverages multi-module collaboration by employing a fine-tuned Llama-3-8b model in conjunction with a collaborative architecture, like a Retriever-Augmenter-Generator (RAG) based context retriever, refiner and dynamic memory of conversation history. SQLAssist demonstrates strong performance in handling simple queries and immediate follow-up questions. A key strength of SQLAssist lies in its unified approach utilizing RAG. By retrieving relevant indexed tables and sample rows, the framework effectively embeds context, enabling the model to generate accurate SQL queries. While the current implementation utilizes a 4-bit quantized Llama-3-8b model with promising results, further performance improvements are expected through exploration of larger models. Additionally, future work will focus on training the model to handle queries involving aliases, joins, and aggregate statements. Benchmarking against established datasets to compare performance with other methods.

Overall, SQLAssist offers a promising foundation for robust Text-to-SQL translation with its collaborative architecture, unified approach to context embedding and history with scope for handling complex queries.

REFERENCES

[1] 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* (2023).

[2] Zefeng Cai, Xiangyu Li, Binyuan Hui, Min Yang, Bowen Li, Binhua Li, Zheng Cao, Weijie Li, Fei Huang, Luo Si, and Yongbin Li. 2022. STAR: SQL Guided Pre-Training for Context-dependent Text-to-SQL Parsing. arXiv:2210.11888 [cs.CL]

[3] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG]

[4] Prajkta Devgade. 2021. The Natural Language Query using NLP by Generation of SQL Query. *International Journal for Research in Applied Science and Engineering Technology* 9 (05 2021), 381–383. https://doi.org/10.22214/ijraset.2021.34145

[5] Chunxi Guo, Zhiliang Tian, Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang. 2024. Retrieval-Augmented GPT-3.5-Based Text-to-SQL Framework with Sample-Aware Prompting and Dynamic Revision Chain. In *Neural Information Processing*, Biao Luo, Long Cheng, Zheng-Guang Wu, Hongyi Li, and Chaojie Li (Eds.). Springer Nature Singapore, Singapore, 341–356.

[6] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. arXiv:1902.00751 [cs.LG]

[7] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 328–339. https://doi.org/10.18653/v1/P18-1031

[8] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL]

[9] Bingchang Liu, Chaoyu Chen, Cong Liao, Zi Gong, Huan Wang, Zhichao Lei, Ming Liang, Dajun Chen, Min Shen, Hailian Zhou, Hang Yu, and Jianguo Li. 2023. MFTCoder: Boosting Code LLMs with Multitask Fine-Tuning. arXiv:2311.02303 [cs.LG]

[10] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]

[11] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions. arXiv:2208.13629 [cs.CL]

[12] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2022. Learning to summarize from human feedback. arXiv:2009.01325 [cs.CL]

[13] Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring Chain-of-Thought Style Prompting for Text-to-SQL. arXiv:2305.14215 [cs.CL]

[14] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242 [cs.CL]