Database Search: Text2SQL using dynamic few-shot prompting with self-consistency using LLM

***Author**: Utkarsh Tripathi and Jeff Shelman, Smart Manufacturing & AI, Micron Technology*

{utripathi, jeffshelman}@micron.com

Abstract

Text2SQL which involves converting natural language to Structured Query Language (SQL) is disruptive application of Large Language Models (LLM). It has potential to radically transform how humans interact with data. This paper proposes a new approach to generate SQL which improves the contextual understanding of LLM significantly. It utilizes a 2-layer dynamic few-shot prompting with self-consistency approach to increase model attention. Also, the redundant information present in few-shot examples is masked and each example is categorized to a respective use-case (domain). Firstly, we store our list of masked few shot examples along with its metadata and vector embeddings in a database. The masking helps in increasing the attention mechanism of LLM to actual context of question instead of irrelevant information present in it. The domain of each few shot example is stored in metadata. Secondly, similarity threshold is also dynamically fetched from database based on the solution where the database search is being used. This dynamic selection of similar few shot examples and similarity threshold based on solution increases the tool flexibility and makes it scalable across different set of tables. Now, when user asks a question, its first categorized into one of the respective use case available in few-shot example set. Once the category of question is finalized, top 5 similar few shot examples from this category are selected based on the similarity score only if the similarity score is above threshold. These few shot examples along with additional instructions are passed to LLM model to generate the SQL query. This step is only performed if the number of similar examples above threshold are at least 2. The cutoff for having minimum number of few shot examples restricts the LLM from hallucinating and keeps model output consistent. Finally, as refinement module, self-consistency is used to select the final answer and SQL query. Adding categorization and masking of irrelevant information in few shot examples increases the accuracy by ~7% resulting in final accuracy of ~94% when tested across 1100 questions of multiple solutions.

Keywords: Gen AI, LLM, Text2SQL, SQL generation, Self-consistency, vector embeddings

1. Introduction

Most of the datasets exists in tabular form stored in one or other relational databases. Extracting insights from these data sources requires deep technical knowledge of writing SQL queries. This restricts the usage of the relational data from broad range of audience who don't have skills to write SQL queries. Also, not every insight from these data sources can be visualized and presented on dashboard. This in turn limits the information accessible to business users which can help in faster decision making.

Text2SQL bridges this gap by making relational database accessible to non-expert users through natural languages. It empowers human to extract valuable information without needing specialized SQL knowledge. Conventional Text2SQL approaches (Li et al., 2023a) often optimize a decoder-encoder network using a quantity of training examples to attain satisfactory Text2SQL outcomes. This fine-tuning paradigm can cause overfitting of the training set and degenerate the transferability of the model. Recent years have seen a significant increase in the use of large language models (LLMs) for Text2SQL. In this context, much of the research has concentrated on using prompting to convert user statements into

SQL queries (Rajkumar et al., 2022; Liu et al., 2023a). More sophisticated prompting techniques have field-specific adaptations to enhance comprehension of natural language inquiries and structured database designs, such as selecting superior few-shot examples based on question similarity (Gao et al., 2023a; An et al., 2023), dividing complex questions into sub-tasks (Tai et al., 2023; Pourreza & Rafiei, 2023), verifying the accuracy of model-generated SQL queries through execution feedback (Chen et al., 2023; Sun et al., 2023; Pourreza & Rafiei, 2023), and connecting NL phrases to appropriate database elements (e.g. the Name column in the County table, see (Pourreza & Rafiei, 2023)).

While these approaches improve Text2SQL performance, some of these require fine tuning the LLM model to a particular dataset and others are not scalable if there are multiple tables present. Real world Text2SQL exhibits a variety of challenges when we scale to multiple tables along with dealing ambiguity present in user's natural language questions. Moreover, real-world databases may contain large volumes of tables and columns, and the sheer content size would easily exceed the context limit of LLMs.

To tackle these obstacles, we suggest a dynamic Retrieval Augmented Generation (RAG) framework that is adaptable to multiple tables and seamlessly integrates with various user interface dashboards, web applications, and other platforms. Our solution employs a dynamic few-shot prompting strategy by storing a substantial collection of Question-Answer (QA) pairs with their corresponding vector embeddings in a table named "qa_pairs" within the vector database. This table serves as a knowledge base for the Large Language Model (LLM), specifically the LLM model. All these questions pertain to diverse analytical solutions and are additionally categorized into specific groups based on the type of question they seek to answer. Both the category and unique identifier of each analytical solution are preserved as metadata associated with these questions. Furthermore, an additional table called "business_rules" is maintained, containing prompt instructions that will be transmitted alongside few-shot examples to the LLM for generating SQL queries. This table stores prompt instructions tailored to different types of analytical solutions, along with their unique identifiers acting as metadata. Beyond the QA pairs and instructional prompts, a distinct "configuration" table is maintained, housing the URL of the analytical solution where Text2SQL is implemented, the unique identifier of the solution, the similarity threshold for filtering few-shot examples, the quantity of few-shot examples to filter above the similarity threshold, the minimum number of examples necessary for the LLM to generate a SQL query and the number of SQL queries to generate for self-consistency. The dynamic nature of this tool enhances its scalability, as no code modifications are required to deploy it for any new solution. The pertinent information merely needs to be inserted into the specified tables, and the tool will operate accordingly. The tool also incorporates a self-consistency approach, generating multiple SQL queries and subsequently conducting a voting process to determine the final SQL query. This chosen SQL query is then executed against a relational database, and the resulting output is extracted. The generated SQL query, along with the extracted results, is subsequently presented to the LLM to produce a summary for the end-user. This summary encompasses an explanation of the SQL query and the generated results. Integrating the Text2SQL solution into any analytical platform empowers business users to swiftly extract valuable insights or uncover hidden information within their underlying data, thereby accelerating decision-making processes and reducing their reliance on specialized SQL knowledge.

2. Preliminaries

2.1 Problem Definition

Given a natural language question Q and a database D, the goal of LLM-based Text2SQL is to transform Q into a functional SQL query Y.

2.2 LLM for Text2SQL

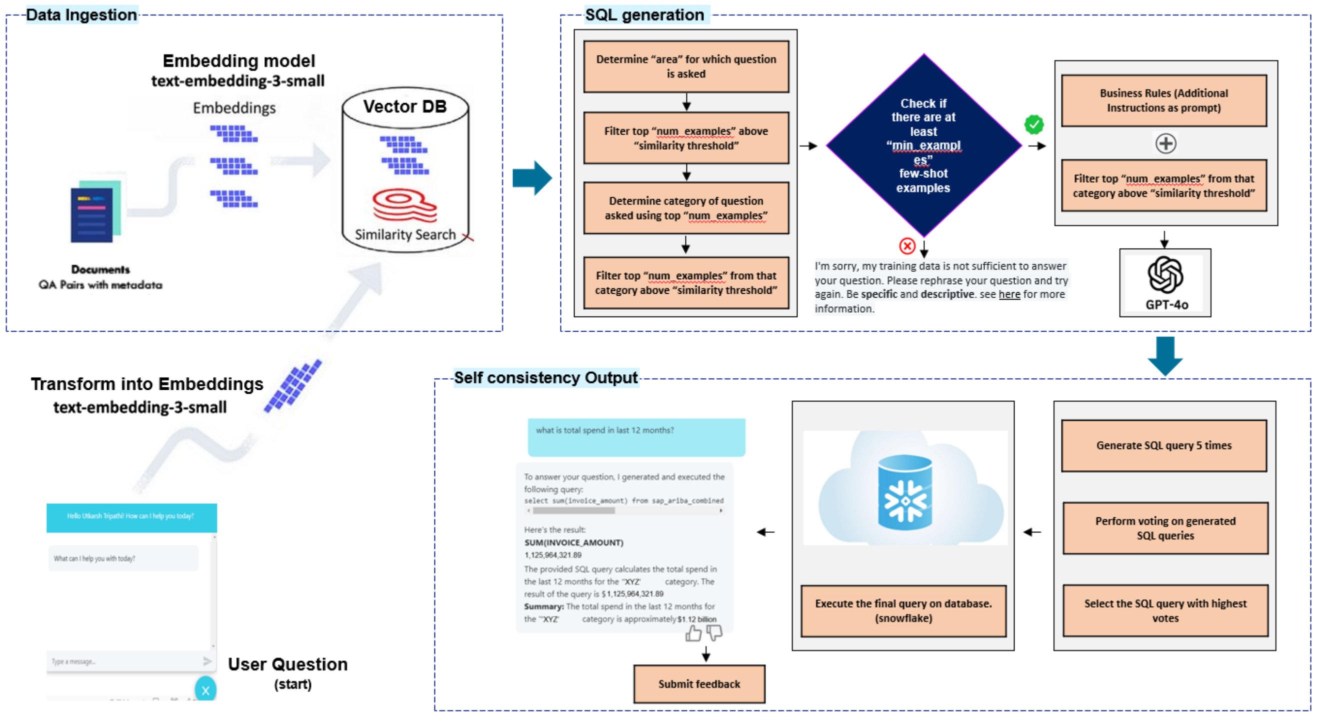
Recent studies (Sun et al., 2023; Liu et al., 2023) have framed the Text2SQL parsing task as a generation problem, employing suitable prompts P to guide a large language model M. This model assesses a probability distribution over SQL queries Y, enabling us to generate queries token by token. The generation process for the SQL query Y can be formulated as follows:

$$P_M\,(Y|P,D,Q) = \sum_{i=1}^{|Y|} P_M\,(Yi|P,D,Q,Y<i) \tag{1}$$

Here, $Y < i$ is the prefix of the SQL query Y and $P_M\,(Yi|\cdot)$ is the conditional probability of the i -th token in the SQL query Y given the prefix $Y < i$, the prompt P, the database D and question Q

3. Proposed Approach

In this section, we will discuss how dynamic few-shot prompting, combined with self-consistency, can aid in generating SQL queries using the LLM model. The framework is illustrated in Figure 1.



Data Ingestion
Embedding model
text-embedding-3-small
Embeddings
Vector DB
Similarity Search
Documents
QA Pairs with metadata
SQL generation
Determine "area" for which question is asked
Filter top "num_examples" above "similarity threshold"
Determine category of question asked using top "num_examples"
Filter top "num_examples" from that category above "similarity threshold"
Check if there are at least "min_examples" few-shot examples
Business Rules (Additional Instructions as prompt)
Filter top "num_examples" from that category above "similarity threshold"
GPT-4o
Transform into Embeddings
text-embedding-3-small
Self consistency Output
Generate SQL query 5 times
Perform voting on generated SQL queries
Select the SQL query with highest votes
Execute the final query on database. (snowflake)
Submit feedback
User Question
(start)

I'm sorry, my training data is not sufficient to answer your question. Please rephrase your question and try again. Be **specific** and **descriptive**. see here for more information.

Hello Utkarsh Tripathi! How can I help you today?

What can I help you with today?

Type a message...

what is total spend in last 12 months?

To answer your question, I generated and executed the following query:

```
select sum(invoice_amount) from sap_ariba_combined
```

Here's the result:

SUM(INVOICE_AMOUNT)

1,125,964,321.89

The provided SQL query calculates the total spend in the last 12 months for the "XYZ' category. The result of the query is $1,125,964,321.89

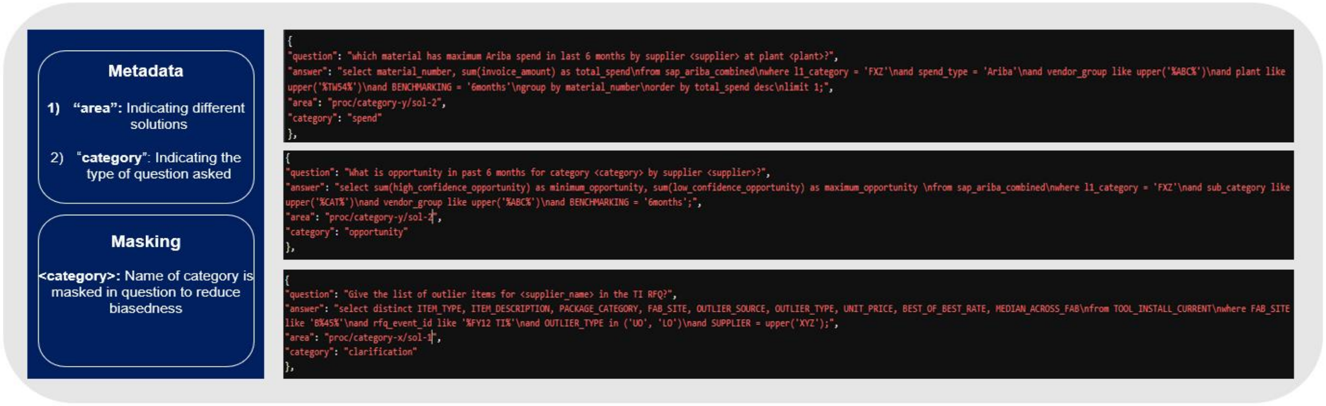
Summary: The total spend in the last 12 months for the "XYZ' category is approximately $1.12 billion

Figure 1

It primarily comprises three components: Data ingestion, SQL query generation through dynamic few-shot prompts, and self-consistency output. The specifics of each component are detailed below.

3.1 Data ingestion (Inserting QA pairs for few shot examples)

In this phase, we construct QA pairs specifically tailored to the analytical solution where the Text2SQL tool will be deployed. These pairs consist of relevant questions and their corresponding correct SQL queries. Each QA pair is categorized under a specific "category" that indicates the use-case the question addresses. In addition to the category, an extra metadata field, "area," is included to specify the solution to which the question pertains. To enhance the LLM's focus on the actual context of the question rather than irrelevant information, certain parts of the question content are masked. For instance, the actual category name is replaced with "<category>" to prevent bias during the similarity check with the user's query as show in figure 2.


Metadata
1) "area": Indicating different solutions
2) "category": Indicating the type of question asked
Masking
<category>: Name of category is masked in question to reduce biasedness

```
{
"question": "which material has maximum Ariba spend in last 6 months by supplier <supplier> at plant <plant>?",
"answer": "select material_number, sum(invoice_amount) as total_spend\nfrom sap_ariba_combined\nwhere l1_category = 'FXZ'\nand spend_type = 'Ariba'\nand vendor_group like upper('%ABC%')\nand plant like upper('%TW54%')\nand BENCHMARKING = '6months'\ngroup by material_number\norder by total_spend desc\nlimit 1;",
"area": "proc/category-y/sol-2",
"category": "spend"
},
```

```
{
"question": "What is opportunity in past 6 months for category <category> by supplier <supplier>?",
"answer": "select sum(high_confidence_opportunity) as minimum_opportunity, sum(low_confidence_opportunity) as maximum_opportunity \nfrom sap_ariba_combined\nwhere l1_category = 'FXZ'\nand sub_category like upper('%CAT%')\nand vendor_group like upper('%ABC%')\nand BENCHMARKING = '6months';",
"area": "proc/category-y/sol-2",
"category": "opportunity"
},
```

```
{
"question": "Give the list of outlier items for <supplier_name> in the TI RFQ?",
"answer": "select distinct ITEM_TYPE, ITEM_DESCRIPTION, PACKAGE_CATEGORY, FAB_SITE, OUTLIER_SOURCE, OUTLIER_TYPE, UNIT_PRICE, BEST_OF_BEST_RATE, MEDIAN_ACROSS_FAB\nfrom TOOL_INSTALL_CURRENT\nwhere FAB_SITE like 'B%45%'\nand rfq_event_id like '%FY12 TI%'\nand OUTLIER_TYPE in ('UO', 'LO')\nand SUPPLIER = upper('XYZ');",
"area": "proc/category-x/sol-1",
"category": "clarification"
},
```

Figure 2

The above QA pairs are passed to embedding model "text-embedding-3-small" and vector embeddings are generated for each of the question present in QA pair. These embeddings along with QA pairs and its metadata are then stored in a table within vector database named "qa-pairs"

Beyond these QA pairs, two additional tables, "business-rules" and "configuration," are maintained. All supplementary instructions that must be transmitted alongside few-shot examples are stored in the "business-rules" table, with "area" serving as the metadata. The "configuration" table houses the URL and "area" of the solution, along with other dynamic information that will be passed to the next step. This dynamic information includes:

a) "similarity_threshold": This specifies the minimum similarity score above which few-shot examples should be filtered from the QA pairs.
b) "num_examples": This indicates the number of similar examples to be passed as few-shot based on the highest similarity score.
c) "min_examples": This represents a cutoff value that determines the minimum number of examples required to generate a SQL query.

3.2 SQL Generation

When a user asks a question about a specific analytical solution, the relevant QA pairs for that solution are filtered from the "qa-pairs" table using metadata stored under the "area" field. The user's question is then processed through the embedding model "text-embedding-3-small" to generate vector embeddings. Using these embeddings, a similarity score is calculated for all questions in the filtered set of QA pairs. The top "num_examples" with scores above the "similarity_threshold" are selected. The scores of these filtered "num_examples" are then aggregated and summed up across the "category" of the filtered examples. The category with the highest summed score is considered the final category for the user's question. Finally, the top "num_examples" from this category, which are above the "similarity_threshold," are chosen as the final few-shot examples for the LLM model.

In addition to the selected few-shot examples, relevant instructions for the analytical solution are filtered from the "business-rules" table based on the "area" metadata. These few-shot examples, the instruction prompt, and the user's question are concatenated to form a final prompt, which is then passed to the LLM model to generate the SQL query. To minimize hallucination and increase questions confidence in the tool's output, the SQL query is generated by the model only if the number of examples above the similarity threshold for the question's category exceeds the "min_examples" threshold.

Number of few shot examples (“num_examples”) >= Minimum number of examples (“min_examples”)

(2)

If the above condition is not satisfied tool will return a message to user saying that “This question cannot be answered since training data is not sufficient. Please try rephrasing the question. Be specific and descriptive”.

3.3 Self Consistency Output

The above-mentioned SQL query generation process is repeated five times to produce five different SQL queries. A voting mechanism is then used to select the final SQL query with the highest votes. This final SQL query is executed on the relational database (Snowflake in this case), and the results are stored. Both the results and the SQL query are then passed to the LLM to generate a summary of the results and an explanation of the generated SQL query. If the results are correct, the user can provide feedback using a thumbs up/thumbs down button. This feedback helps in collecting more, which can later be stored in the "qa-pairs" table to expand the knowledge base.

In case the generated SQL query is syntactically wrong, the error is passed to LLM to summarize it which explains the error and return a possible correct SQL query which can be analyzed, and same question can be stored later in knowledge base.

Since self-consistency is used to generate 5 SQL queries, the temperature of LLM model is kept moderately high (temperature = 0.3) to create some variations in generated SQL queries. This is done so that voting can be utilized efficiently to select final SQL query.

4. Results

Combining all elements, the proposed approach achieves high accuracy for the questions tested across two different analytical solutions. By masking irrelevant information in few-shot example questions and categorizing each into specific use cases, the accuracy of the approach improved significantly. We compared the results of approach without masking and categorizing the few-shot examples into “category” with masking and categorization of few-shot examples. The analysis of accuracy of solution across different use cases (“category”) within each solution (“area”) is presented in detail in Table 1. Putting a cutoff of having minimum number of examples as described in (2) resulted in some questions being non-answerable with response “This question cannot be answered since training data is not sufficient. Please try rephrasing the question. Be specific and descriptive”. This is shown under column “Not enough examples*” of Table 1. Providing no response instead of wrong output increases the trust on proposed approach since its deployed-on top of analytical solutions which provides actionable insights to the business users. This resulted in an overall accuracy of ~94.7% across 2 tested solutions. The individual accuracy for each solution and its respective “category” is show in table below.

Area	Category	Number of Questions	Correct Answers	Not Enough Data for training*	Wrong Answer	Accuracy	Overall Accuracy
proc/category-x/sol-1	Spend	176	160	14	2	90.9%	93.39%
	Negotiaion	190	181	7	2	95.3%	
	Clarification	300	281	18	1	93.7%	
proc/category-y/sol-2	Spend	57	55	2	0	96.5%	95.97%
	Opportunity	65	60	0	5	92.3%	
	Arbitrage opportunity	150	144	0	6	96.0%	
	Trend Opportunity	150	146	0	4	97.3%	

Table 1

5. Conclusion

This paper presents a novel 3-step holistic framework (RAG architecture) leveraging pre-trained LLMs to address the challenges inherent in Text2SQL tasks. The framework utilizes a dynamically selected set of few-shot examples from a vector database, combined with self-consistency, to generate accurate SQL queries. To further enhance performance, we propose masking irrelevant information within these few-shot examples and categorizing them into specific "Categories." This approach allows the LLM to focus on the essential context of the question while ensuring that the few-shot examples originate from a defined set, mitigating the risk of providing inaccurate examples. Our methodology resulted in a significant accuracy improvement, raising the solution accuracy from approximately 88% to 94.7%, a substantial increase of roughly 7%. Recognizing the crucial need for user trust when deploying the solution on tables providing actionable insights, we implemented a mechanism to restrict the LLM from generating output if the number of filtered few-shot examples falls below a predetermined threshold. This dynamic approach makes the framework scalable and readily deployable on any new database. Overall, the proposed framework demonstrates promising results, opening new avenues for advancements in Text2SQL tasks.

6. Future work

The primary hurdle in this endeavor lies in the creation of a dataset containing pertinent few-shot examples, a task that demands significant domain expertise. The challenge of incorporating domain knowledge into models and ensuring their efficient deployment across diverse domains remains a critical issue, particularly when dealing with domains that share similar information stored in databases but exhibit slightly divergent database schemas.

Instead of relying solely on self-consistency, a cross-consistency approach could be explored by integrating several state-of-the-art large language models and ultimately selecting the final output through a voting mechanism. Furthermore, enhancing the accuracy of this approach could be achieved by fine-tuning the large language model through the provision of a data dictionary encompassing tables and their associated column descriptions.

7. Acknowledgement

We also extend our gratitude to Li Ye and Niladri Sekher Karmakar from the SMAI business function team for their valuable suggestions during the development of the solution.

References

Li, H., Zhang, J., Li, C., & Chen, H. (2023). RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. Proceedings of the AAAI Conference on Artificial Intelligence, 37(11), 13067-13075. https://doi.org/10.1609/aaai.v37i11.26535

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. arXiv preprint arXiv:2204.00498.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. arXiv preprint arXiv:2308.15363, 2023a.

Shengnan An, Bo Zhou, Zeqi Lin, Qiang Fu, Bei Chen, Nanning Zheng, Weizhu Chen, and Jian-Guang Lou. Skill-based few-shot selection for in-context learning. arXiv preprint arXiv:2305.14210, 2023.

Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. Exploring chain-of-thought style prompting for text-to-sql. arXiv preprint arXiv:2305.14215, 2023.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. arXiv preprint arXiv:2304.11015, 2023.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. arXiv preprint arXiv:2304.05128, 2023.

Ruoxi Sun, Sercan Ö Arik, Rajarishi Sinha, Hootan Nakhost, Hanjun Dai, Pengcheng Yin, and Tomas Pfister. Sqlprompt: In-context text-to-sql with minimal labeled data. arXiv preprint arXiv:2311.02883, 2023

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. arXiv preprint arXiv:2303.13547, 2023a