Large Language Model Based SQL Generator (LLSQ)

PODDUTOORI NEETHA REDDY
Dept. computer science and engineering
Anurag University, Hyderabad, Telangana, India

MADASU SURYA TEJA
Dept. computer science and engineering
Anurag University, Hyderabad, Telangana, India

Abstract: - In the contemporary landscape, accessing databases poses challenges for untrained individuals due to technical complexities. Natural Language to SQL (Structured Query Language) generator models have alleviated these obstacles but struggle with extensive databases, often leading to inaccuracies stemming from inadequate training data. A proposed solution introduces a novel model merging OpenAI models with Langchain technology. This innovative approach aims to overcome limitations by harnessing OpenAI's language capabilities alongside Langchain's rapid processing, enabling efficient handling of vast database information. By leveraging OpenAI's advanced large language models and Langchain's high-speed processing, this model seeks to bridge the gap for non-technical users, swiftly converting natural language queries into precise SQL commands. Its strength lies in accurately interpreting queries even with massive datasets, promising enhanced database accessibility for individuals lacking technical expertise. This innovation could foster inclusivity and user-friendly data utilization by making databases more accessible.

Keywords:- Natural language, SQL, Langchain technology, OpenAI, queries

I. INTRODUCTION

SQL (Structured Query Language) queries are commands used to interact with relational databases. These queries are designed to retrieve, manipulate, and manage data stored in a database. SQL is a standard language for interacting with relational database management systems (RDBMS), and it provides a consistent way to perform various operations on databases.

In the continuously changing landscape of technology, the demand for efficient and accessible interfaces to interact with databases has become increasingly perceivable. Traditional methods of querying databases through Structured Query Language (SQL) require a certain level of expertise, hindering accessibility for individuals without a technical background. The challenges faced by Natural Language to SQL generator models, particularly in handling extensive databases, have been a point of contention. These models, designed to simplify database interactions, often encounter inaccuracies due to limitations in their training data. Anisha T S et al. [3] established a model using deep learning to provide SQL queries which leads to the performance of the model that heavily relies on the quality and size of training data, potentially leading to biases or limited applicability to specific domains.

Existing methods often compromise on accuracy and struggle with managing large databases [2] accuracy dependence on a well-defined and accurate database schema for accurate SQL generation.To address these issues, the integration of OpenAI models with Langchain technology to generate SQL Query.

OpenAI's large language models, such as GPT-3, excel at understanding and generating human-like text. This capability is crucial for interpreting natural language queries input by users. The generator can utilize OpenAI's models to comprehend the meaning, context, and intent behind user queries, even when expressed in colloquial or non-standard language.OpenAI can assist in the translation of natural language queries into SQL commands. By training or fine-tuning the language model on SQL-related tasks, the generator can effectively convert user-friendly language into the precise syntax required by the SQL database. This translation process is essential for ensuring that the generated SQL queries accurately reflect the user's intent.

The integration of OpenAI enhances the performance of Natural Language to SQL generators by combining the capabilities of large language models with the advancements brought by Langchain technology. This merger aims to overcome the challenges posed by extensive databases, providing a more robust and accurate solution for translating natural language queries into SQL commands. The synergy between OpenAI's large language model understanding capabilities and the specialized features offered by Langchain technology seeks to improve the overall accuracy and effectiveness of the generator, ensuring a more seamless and reliable user experience.

II. LITERATURE SURVEY

A.Deshpande et. al. [1] proposed a model aiming to bridge the gap in SQL expertise among users interacting with databases. The methodology involves Natural Language Processing (NLP), and its component employs either pre-defined rules or trained models to map natural language to SQL constructs. However, this NLP technology relies on a well-defined and accurate database schema for accurate SQL generation. T S, Anisha et al. [2] have explored the application of deep learning in text-to-SQL conversion, aiming to automatically translate natural language questions into SQL queries within NLP. The authors conducted research, using common deep learning techniques like Sequence-to-Sequence(Seq2Seq) Models. Training and evaluation were based on datasets containing natural language queries and corresponding SQL queries.The main challenges faced in this research are handling complex queries, generalizing to unseen databases, and ensuring robustness to noise and ambiguity in natural language. K. Lahoti et al. [3] have explored deep learning approaches for text-to-SQL semantic parsing, including Sequence-to-Sequence Models and Structured Prediction Models. These methods are utilized for encoding-decoding language, breaking down SQL queries, and pre-training large models. However, achieving high accuracy in translating complex natural language queries into correct SQL remains challenging. A. Kate et al. [4] address the automatic translation of natural language questions into executable SQL queries. Rule-based systems are used for predefining the rules for constructing the SQL and machine learning models for training datasets of natural language queries and corresponding SQL queries to learn the translation patterns. M. Arefin et al.[5] developed a model that automatically translates natural language questions into executable SQL queries. They incorporated supervised learning for data collection, preprocessing, model selection, training, and evaluation. However, they struggled with the availability of quality datasets for training, which were crucial but limited in size and diversity. P. Parikh et al. [6] address the challenges associated with SQL by creating an application that transforms natural language questions from course instructors into SQL queries. This is achieved through the implementation of a Seq2Seq model, which generates SQL queries to retrieve relevant data displayed on a dashboard through an e-learning domain. X. Xu et al. [7] opted for supervised learning that employs Structured prediction with Sketch-Based decoding, breaking down SQL query generation into structured decisions using sketches for syntactic correctness and reduced search space. The method faces scalability problems. T. Shi et al.[8] the incremental Text-to-SQL model generates SQL queries, employing a Seq2Seq model with attention or a structured prediction approach, enabling context-aware decisions and potential revisions during query generation. It undergoes non-deterministic Oracle training, learning from examples where multiple correct SQL queries are possible for a given input. This approach teaches the model to handle ambiguity, encouraging the production of diversity. But this approach requires a lot of training time. C. Wang et al. [9] employed an iterative decoding with execution feedback, generating SQL queries in multiple steps and interacting with the database at each stage. Error correction and refinement mechanisms detect and address potential errors in generated queries based on execution feedback, enhancing the chances of successful execution.Integrating execution feedback into the decoding process adds complexity to model design and training. Moses Visperas et. al.[10] have come up with an idea of text-to-SQL semantic parsing within Natural Language Interfaces to Databases (NLIDBs), likely conducting a comprehensive review and comparison of recent approaches. They stated the need for high accuracy in translating diverse natural language queries to SQL and the challenge of explaining the reasoning behind generated SQL queries, posing difficulties in debugging and addressing user concerns. C. Wang et. al. [11] This model aims to generate diverse and complex SQL queries, surpassing the capabilities of many text-to-SQL systems, utilizing Neural Machine Translation, Structured Prediction, and Grammar Induction methods. The approach enables the generation of intricate queries and the learning of new structures and patterns from additional examples for enhanced flexibility. M. Brockschmidt et.al.[12] developed automatically identifying and extracting SQL queries embedded within natural language text, separating them from surrounding text. In this model the analysis approaches are used, using these approaches parsing of the queries and analysis of query semantics is done to confirm alignment with the surrounding text context. N. Yaghmazadeh et. al.[13] have developed a translation of natural language questions about a database into executable SQL queries. Mapping Stage is used for establishing connections between natural language and database structures. Sqlizer dynamically generates a structured SQL query that retrieves the information requested in the natural language question. However, models trained on specific databases might not perform well on unseen database structures. Po-Sen Huang et al.[14] methodology employs meta-learning to generate structured queries from natural language by treating each example as a unique task, creating pseudo-tasks, and using a meta-learning algorithm like MAML for quick adaptation. J. M. Zelle et al.[15] have observed that logical inductive programming involves providing a knowledge base (KB) with rules about database query structure, encompassing both general syntactic rules (e.g., SQL syntax) and domain-specific rules. The learning process utilizes training examples, pairing database queries with correct parse tree representations, as the ILP algorithm iteratively searches for logical rules.When parsing new queries, the learned rules are applied to infer logical structures and generate parse trees for queries. It consumes a lot of time to train the model. B. Bogin et. al.[16] have come up with a methodology involving Schema Graph Construction, representing the database schema as a graph where nodes depict tables, columns, and attributes. A Graph Neural Network (GNN) is then applied to learn representations of each graph node, capturing structural and semantic information, including relationships and dependencies. In Text-to-SQL parsing, the learned schema graph representations are integrated into the model to inform query generation, guiding the process toward producing SQL queries aligned with the database structure. C. Sugandhika et.al.[17] have proposed a Heuristics Application phase that employs predefined rules to map natural language elements to SQL constructs, addressing specific patterns such as mapping nouns to tables and handling aggregations. SQL Query Generation assembles components based on heuristics, ensuring syntactic correctness and schema adherence. Heuristics might not always capture the correct intent of the user's query. A. Anisyah et. al[18] have developed a Natural language interface to bridge the gap between natural human language and structured database queries, empowering users to interact with databases without requiring technical SQL knowledge. Xiaoyu Zhang et. al.[19] have proposed the Multi-task SQL (M-SQL), leveraging a pre-trained BERT model for semantic understanding. It integrates modules for value extraction and column matching, identifying query values and associating them with relevant table columns. The joint decoder utilizes BERT's encoded representation and outputs from extraction modules to generate the final SQL query. I.Androutsopoulos et.al.[20] have proposed an SQL that operates through three stages: understanding the query using a user-defined knowledge base, generating an SQL query in a Logical Query Language (LQL), and refining the query for errors and optimization. It poses an issue with potential performance overhead for large databases. Muhammad Shahzaib Baiget et al.[21] review the translating natural language queries into SQL compares existing frameworks based on aggregation classifier, select column pointer, and clause pointer, emphasizing the role of semantic parsing and neural algorithms in predicting these components. B. Sujatha et al.[22] EFFCN (Efficiently Compliant Flexible Compliant Natural language interface to a database) involves parsing the query to identify key elements, mapping this information to SQL clauses using rules or templates, and refining the generated SQL. Potential reliance on handcrafted rules may limit its ability to handle unseen or complex queries. Abhishek Kharade et al.[23] generate SQL queries using Natural Language Processing and review encompasses existing approaches, ranging from rule-based systems to advanced machine learning models, highlighting diverse methodologies in the field. It concludes with the performance issues and lack of accuracy in the existing methods. Adrián Bazaga et al.[24] model adopts a "polyglot" approach for translation, allowing it to generate database specific SQL for various engines. It uses a rule-based parser to extract key information. The model performance depends on the quality and size of the training data. Pwint Phyu et al. [25] Use NLP on database queries allowing users to interact with databases using plain English. This process involves understanding the user's question, interpreting the semantic meaning to match the database schema, and then generating and executing an SQL query. T. H. Y. Vuong et al.[26] In the initial phase, specific patterns are utilized to recognize components like the SELECT clause and assign them appropriate labels. In the subsequent phase, relevant keywords and entities are extracted for each labeled component, resolving any ambiguities along the way. Extracted values are then inserted into the corresponding slots within the SQL clauses. Finally, these filled slots and clauses are merged to produce the comprehensive SQL query. G. Rao et al.[27] translation to SQL involves establishing formal grammar for sentence structures and word combinations forming semantic parser interpretation. This semantic representation is then matched to the database schema, aligning entities with tables and columns. This process generates a valid SQL query, enabling retrieval of desired data from the database. P. Pasupat et al.[28] logical form grammar for NLP-based database queries focuses on constructing logical forms that match user queries with database tables, through a process of parsing the query, checking types against the table schema, finding the best matching logical form, and generating an answer. A. Pagrut et al.[29] automated SQL query generator that employs NLP to understand user input, matching it with database elements and discerning conditions and operators. It then constructs a valid SQL query, it validates the query, allowing users to review and adjust it as necessary, ultimately executing it on the database and presenting the results. It often requires customization for specific domains and database schemas. R. Kumar et al.[30] The method involves analyzing the user's natural language query through tokenization and morphological analysis, identifying word forms and meaningful phrases. A pattern library links these language structures to SQL constructs via a pattern matching algorithm, to match for elements like tables, columns, and operators. The resulting SQL query is executed on the database.

III. PROPOSED METHOD

The proposed approach has demonstrated a notable improvement in the accuracy of the Logical form, indicating that the generated query accurately captures the content of the entire input. Additionally, it overcomes the limitations of working with extensive databases and generating multi-column queries. The procedure consists of two independent stages: the first stage is fine-tuning the model, and the second stage is using the fine-tuned model to generate queries.

A. Fine-tuning the model

Fine-tuning improves the performance of models that are available via the API by providing better outcomes than prompting, allowing for training on a greater number of examples than those limited by prompts, saving tokens with succinct prompts, and facilitating queries with reduced latency.

The following steps are included in the fine-tuning process:

1. Prepare and upload training data:

The dataset must have the same conversational structure as the Chat Completions API. This implies that it ought to consist of a series of messages, with each message having its own content and a role (such as the speaker). This can only occur if the dataset is converted to a JSON lines file type.

For example:

{"messages": [{"role": "system", "content": "You are an SQL generation expert tasked with creating a system that generates SQL queries. "},

{"role": "user", "content": schema and question},

{"role": "assistant", "content": SQL query}]}

The assistant will get instructions from the system regarding how to handle the data that the user has given it.

We created the dataset in JSON lines format as shown in Fig. 2.

```
messages=[
  {"role": "system", "content": "You are an SQL generation expert tasked with creating a system that generates SQL queries."},
  {"role": "system", "content": "The system takes as input a relational database schema and a natural language question relate
  {"role": "system", "content": "The goal is to translate the user's question into a valid SQL query that extracts the relevan
  {"role": "system", "content": "Inputs:\n1. Schema: The relational database schema provided by the user. The schema includes
  {"role": "system", "content": "Output: The system is expected to generate a valid SQL query based on the provided schema and
  {"role": "user", "content": "Voters (VoterID, FirstName, LastName, Age) <br> Elections (ElectionID, ElectionName, ElectionDa
  {"role": "user", "content": "Retrieve the names of voters who have not cast any votes in a specific election."}
 ]
)
```

Fig. 2. JSON line document

The training data must then be uploaded via the Files API. This makes it possible to use the data for fine-tuning tasks efficiently. Apply the subsequent steps:

Step-1: Import the OpenAI module.
Step-2: Set up an OpenAI client instance for communication.
Step-3: Use binary read mode to access the JSON Lines dataset.
Step-4: Upload the opened file to a new file using the OpenAI client.
Step-5: Set the purpose of the file as "fine-tune."

2. Create a fine-tuned model:

After uploading the file, the next action is to initiate a fine-tuning job. To achieve this, use the OpenAI SDK to create a fine-tuning job:

Step 1: Set up an OpenAI client instance for communication.

Step 2: Create a fine-tuning job using the OpenAI client.
Step 3: Enter the training file ID (such as "file-abc123") that was returned when the training file was uploaded to the OpenAI API as its identifier.
Step 4: Select "gpt-3.5-turbo" as the model to be fine-tuned.

When initiating a fine-tuning job, the completion time can vary due to queue dynamics and training factors such as model and dataset size. Once the training concludes, users receive a confirmation email.

Upon successful completion, the fine-tuned model field in the job details will be populated with the name of the specific model.

The overall algorithm for fine tuning:-

Step 1: Import pandas
Step 2: Upload the CSV dataset
Step 3: Encode the dataset with 'cp1252'
Step 4: Create a new JSONL file named 'training.jsonl' and open it in writing mode
Step 5: Create functionCREATE_DATASET which takes a schema, question and its corresponding query as the input parameters.
Step 6: The function creates a message as shown in the above example and returns it.
Step 7: Iterate through the CSV file and send the schema, question, and query one by one to CREATE_DATASET.
Step 8: While iterating write the returned message (every time in a new line) from CREATE_DATASET in training.jsonl.
Step 9: From the 'openai' library import the 'OpenAI' class
Step 10: Create an instance of the OpenAI class as 'client' with an 'api_key' parameter set as user API KEY, which will be used to communicate with the OpenAI API.
Step 11: Upload the training file using 'file. create()' method under 'OpenAI' and set its 'file' parameter as open 'training.jsonl' file in read binary mode and set the 'purpose' parameter as 'fine-tune'. A file ID will be generated.
Step 12: Fine-tune the model using fine_tuning.jobs.create()' method under the OpenAI class and set its parameters. Set 'training_file' as the file id generated after uploading the training file and set the 'model' as the 'gpt-3.5-turbo'. This will create a fine-tuning job id.
Step 13: Retrieve the fine-tuned model status by using the fine_tuning.job.retrieve() method under OpenAI. The model name will be generated.

B. Generating SQL queries using the fine-tuned model:

1. *Install Python libraries*

In the first stage of the workflow, we install three Python libraries:-
i) OpenAI: Incorporated to utilize the robust language models provided by OpenAI for the purpose of generating queries.
ii) LangChain: Crucial for the efficient transformation of the natural language to SQL query generation.

2. *Launch the OpenAI LLM*

To Utilize advanced language models for SQL query generation, follow the describe process:

Step 1: Import the required libraries from langchain.chat_models module, including OpenAI for interacting with the OpenAI API and ChatOpenAI.
Step 2: Define API key for authentication with the OpenAI API.
Step 3: Build an instance of the ChatOpenAI class with parameters such as the desired model, temperature, maximum tokens, and API key.

3. *Obtain details of the table*

Retrieving basic details about tables in the database. The function executes a SQL query against the database to retrieve basic details about tables in the schema. The query selects three columns from the 'information_schema.columns' view: 'table_name', 'column_name' and'data_type'. The SQL query is executed using the 'cursor.execute()' method with the cursor object passed to the function as an argument. The 'fetchall()' method retrieves all the rows returned by the query. The function returns these fetched rows, which represent basic details about tables in the schema.

4. *Retrieve information about the foreign key*

Retrieve information about foreign keys in the database. A function constructs a SQL query that is stored in the variable 'query_for_foreign_keys'. This query retrieves information about foreign keys from the system catalog tables 'pg_constraint' and 'pg_attribute'. The query is executed using the ‘cursor.execute()’ method with the cursor object passed to the function as an argument. ‘cursor.fetchall()’ retrieves all the rows returned by the query. The function returns the fetched rows, which represent information about foreign keys in the database.

5. *Create vector representations*

The purpose of creating vector representations is to encode and represent data from a Chroma vector database in a format that is suitable for computational analysis and processing.

Step-1: Initialize the CSVLoader object
Step-2: Load the data from the CSV file
Step-3: Chroma vector database is to be created for vector representation.

6. *Save the details related to the database schema*

It is to ensure the structured storage and management of database metadata, particularly pertaining to table information and foreign key relationships.
Step-1: Generate a unique identifier
Step-2: Establish the database connection
Step-3: The information regarding the tables retrieved is stored in the data frame which has to be saved as a CSV file.
Step-4: The foreign key details obtained are to be stored in the data frame and are to be saved as a CSV file.
Step-5: Develop the vector representations

7. *Generate SQL query*

Step 1: Process the input provided by the user.
Step 2: Load table information from the CSV file generated and iterate through each table in the data frame.

Step 3: Chroma vector search is performed to retrieve the related tables based on the input.
Step 4: Process the relevant tables from the data frame.
Step 5: Relevant foreign keys are processed according to the user input.
Step 6: Interact with the large language models and generate the query.

8. Establishing database connection

The connection with the database is developed to access the database and retrieve the information.

Step-1: Connection is developed with the URI of the database provided by the user.
Step-2: Save details related to the database

9. Executing the provided SQL query

Step-1: Connection with the database is established.
Step-2: Parse through the provided query
Step-3: The parsed and extracted query is executed against the database server.
Step-4: The response of the executed query against the database is displayed.

The overall algorithm of SQL query generation:

Step 1: Install OpenAI, LangChain, and Psycopg2
Step 2: Import the OpenAI library and assign the user's API key to the '; API_KEY' variable.
Step 3: From the 'OpenAI' library, import the 'ChatOpenAI' class.
Step 4: Initialize an instance of the ChatOpenAI class as 'llm' with parameters. Set 'temperature' to 0 to keep the output more focused and less random, set 'max_tokens' to 1000 to process the input within that limit, set 'openai_api_key' to API_KEY to access the OpenAI API,
and set 'model' to the model name obtained after fine-tuning.
Step 5: Define a function that takes the ‘cursor’ as its input parameter. The function retrieves basic information about tables and their columns from a PostgreSQL database using SQL queries and the results are fetched using the cursor's 'fetchall' method.
Step 6: Create a function that takes ‘cursor’ as an input parameter essentially retrieves information about foreign key constraints in the database and returns it in a structured format, typically as a list of tuples or a similar data structure. It fetches the results using the cursor's 'fetchall' method.
Step 7: Define a function ‘create_vectors’ that takes two arguments: ‘filename’ (the path to the CSV file) and ‘persist_directory’ (the directory where the generated vectors will be stored).
Step 8: Import the ‘CSVLoader’ class from document_loaders and csv_loaders modules from the Langchain package for loading the data from the CSV files.
Step 9: Import the ‘Chroma’ class from the vectorstores module from the long-chain package for the vector representation of the CSV files that are created by using chroma.
Step 10: Initialize 'Chroma' object 'vectordb'. Use the 'from_documents' method of the Chroma object to create vectors from the loaded data (data). It takes the loaded data as input, along with some additional parameters such as embedding (presumably

a method for creating embeddings) and persist_directory

Step-11: Import the 'uuid4' class from the uuid package to generate a unique ID for every CSV file and vector representation.

Step-12: Import the pandas package to store the details about the database in the data frame.

Step-13: Import the 'ChatPromptTemplate' class from the langchain package.

Step-14:Import'HumanMessagePromptTemplate' class from the chat module of the langchain package, for processing the user input.

Step-15:Import'SystemMessagePromptTemplate' class from the chat module of the langchain package, for generating the responses to the user input.

Step-14: Construct a SQL query function for generating a query that takes relevant_tables, table_info, and foreign_key info as its parameters and stores the result in an 'answer. content' variable.

Step-15: Import the 'Psycopg2' package to establish a connection with the database server.

Step-16: Construct an execute query method that takes the generated query and database uri as its parameters for executing the query on the provided database server. The result obtained is stored in the 'result' variable.

IV. RESULTS AND DISCUSSION

In order to evaluate the efficacy of system-generated queries, we conducted experiments with five distinct models, GNN [16], BERT [19], Semantic Parser [27], Seq2Seq [2], Rule-Based Generation [24] and our model- LLSG (Large Language model based SQL Generation) to assess their performance. Our evaluation entails the presentation of results Precision Score, Execution Accuracy and Semantic Accuracy.

1. Precision Score

Precision score refers to the ratio of correctly generated SQL queries to the total number of generated queries. It measures the accuracy of the generated queries. A high precision score indicates that the generator is generating accurate SQL commands in response to natural language queries, minimizing the chances of inaccurate queries.

$$Precision\,Score\ =\ \frac{Number\ of\ correctly\ Generated\ Queries}{Total\ Number\ of\ Queries}$$

Table I depicts the average precision scores of various SQL query generation techniques applied to 50 different SQL queries. Our findings show that LLSG performs better than other approaches, with a precision score of 0.894.

Table I: Average Precision Score of Various Models

Models	**Average Precision Score**
GNN	0.482
BERT	0.393
Semantic Parser	0.710
Seq2Seq	0.241
Rule Based Generator	0.358
LLSG	0.894

The precision score percentages for all six models are illustrated in Fig. 1.

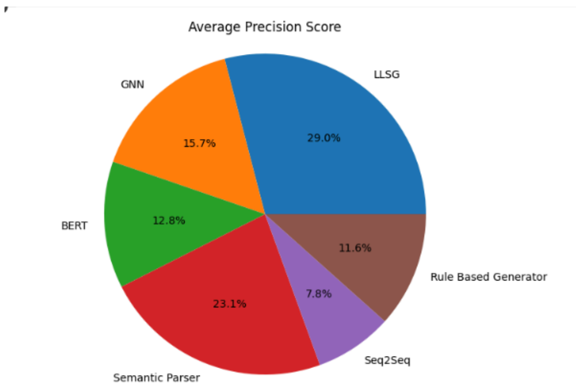

Fig. 1. Precision Score of Models

2. Execution Accuracy

Execution Accuracy evaluates the generated SQL queries for their correctness in terms of executing against the database. Incorrect queries may lead to errors or undesired outcomes. This is crucial as it assesses whether the generated queries are syntactically and semantically correct and whether they retrieve the desired information from the database.

$$Execution\ Accuracy = \frac{No.\, of\ correctly\ executed\ queries}{total\ no.\ of\ queries} \times 100$$

Table II displays the Execution Accuracy for various models applied to a given set of generated queries. Examining the table, it is evident that the LLSG exhibits the highest Execution Accuracy.

Table II: Average Execution Accuracy of Different Models

Models	**Average Execution Accuracy**
GNN	0.71
BERT	0.50
Semantic Parser	0.39
Seq2Seq	0.68
Rule-Based Generator	0.24
LLSG	0.92

The Execution Accuracy percentages for all six models are illustrated in Fig. 2.

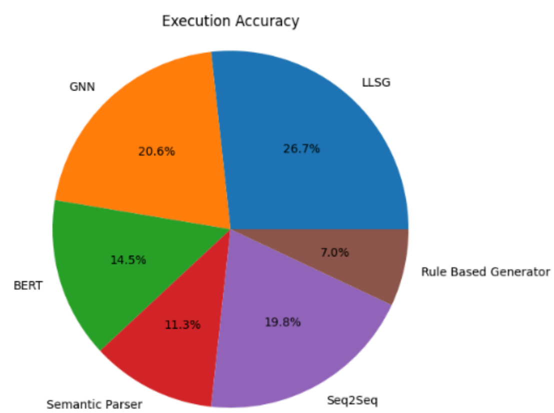

Fig.2. Execution Accuracy of Models

3. Semantic Accuracy

Semantic accuracy assesses the extent to which the generated SQL queries convey the correct meaning or intent of the input natural language queries. Semantic accuracy helps ensure that the generated queries are not only grammatically correct but also functionally correct in terms of their intended purpose.

$$Semantic\ Acc = \frac{No.\ of\ Queries\ With\ Correct\ Semantic\ Capture}{Total\ No.\ of\ Queries} \times 100$$

Table III displays the Semantic Accuracy for various models applied to a given set of generated queries. Examining the table, it is evident that the LLSG exhibits the highest Semantic Accuracy.

Table III: Average Semantic Accuracy of Various Models

Models	Average semantic accuracy
GNN	0.41
BERT	0.60
Semantic Parser	0.84
Seq2Seq	0.32
Rule-Based Generator	0.29
LLSG	0.90

The Execution Accuracy percentages for all six models are illustrated in Fig. 3.

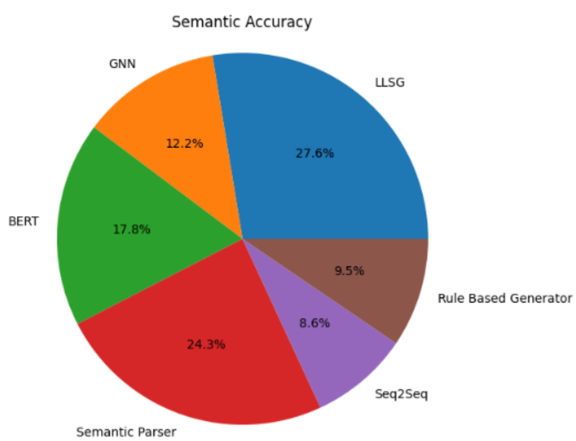

Fig.3. Semantic Accuracy of models

V. CONCLUSION

The research has illustrated that the LLSG for SQL query generation distinguishes itself by perfectly encapsulating the essence of natural language. In contrast to other models that face difficulties in producing executable queries and struggle to capture the semantics of natural language, the proposed method effectively addresses these issues. Through the implementation of the meticulous fine-tuning of OpenAI's language model with a robust dataset, the proposed method excels in providing comprehensive and conscientious SQL queries, exhibiting its effectiveness in improving SQL generation.

VI. REFERENCES

[1] A. Deshpande, D. Kothari, A. Salvi, P. Mane and V. Kolhe, "Querylizer: An Interactive Platform for Database Design and Text to SQL Conversion," 2022 International Conference for Advancement in Technology (ICONAT), Goa, India, 2022, pp. 1-6, doi: 10.1109/ICONAT53423.2022.9725828.

[2] T S, Anisha and P C, Rafeeque and Murali, Reena "Text to SQL Query Conversion Using Deep Learning: A Comparative Analysis", In proceedings of the International Conference on Systems, Energy & Environment (ICSEE) 2019, GCE Kannur, Kerala, July 2019

[3] K. Lahoti, M. Paryani and A. Patil "Deep Learning Based Text to SQL Conversion on WikiSQL Dataset: Comparative Analysis", 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques , Ghaziabad, India, 2022, pp. 1-6, doi:10.1109/ICICT55121.2022.10064556.

[4] A. Kate, S. Kamble, A. Bodkhe and M.Joshi "Conversion of Natural Language Query to SQL Query", 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2018, pp. 488-491, doi: 10.1109/ICECA.2018.8474639.

[5] M. Arefin, K.M. Hossen and M. N. Uddin "Natural Language Query to SQL

Conversion Using Machine Learning Approach", 2021 3rd International Conference on Sustainable Technologies for Industry 4.0 (STI), Bangladesh, 2021, pp. 1-6, doi: 10.1109/STI53101.2021.9732586

[6] P. Parikh et al., "Auto-Query - A simple natural language to SQL query generator for an e-learning platform," 2022 IEEE Global Engineering Education Conference (EDUCON), Tunis, Tunisia, 2022, pp. 936-940, doi: 10.1109/EDUCON52537.2022.9766617.

[7] Xu, Xiaojun & Liu, Chang & Song, Dawn. (2017). “SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. “

[8] T. Shi & Tatwawadi, Kedar & Chakrabarti, Kaushik & Mao, Yi & Polozov, Oleksandr & Chen, Weizhu. (2018),“IncSQL: Training Incremental Text-to-SQL Parsers with Non-Deterministic Oracles”.

[9] C.Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov and Rishabh Singh. “Robust Text-to-SQL Generation with Execution-Guided Decoding.”

[10] Moses Visperas, Aunhel John Adoptante, Christalline Joie Borjal, Ma. Teresita Abia, Jasper Kyle Catapang, Elmer Peramo,“On Modern Text-to-SQL Semantic Parsing Methodologies for Natural Language Interface to Databases: A Comparative Study”, 2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pp.390-396, 2023.

[11] C.Wang& Cheung, Alvin & Bodik, Rastislav. (2017),“Synthesizing highly expressive SQL queries from input-output examples”, pp.452-466, doi: 10.1145/3062341.3062365.

[12] Marc Brockschmidt, C.Wang and Rishabh Singh. “Pointing Out SQL Queries From Text.” (2018).

[13] N.Yaghmazadeh, Navid & Wang, Yuepeng & Dillig, Isil & Dillig, Thomas. (2017),“SQLizer: query synthesis from natural language”, vol. 1, pp.1-26, doi: 10.1145/3133887.

[14] Po-Sen Huang, Chenglong Wang, Rishabh Singh and Wen-tau Yih "Natural Language to Structured Query Generation via Meta-Learning", 2018.

[15] J. M. Zelle and R. J.Mooney, “Learning to parse database queries using inductive logic programming,” in Proc. 13th Nat. Conf. Artif. Intell., vol. 2, 1996, pp. 1050–1055.

[16] B. Bogin, M. Gardner, and J. Berant, “Representing schema structure with graph neural networks for Text-to-SQL parsing” in Proc. 57th Annu. Meeting Assoc. Comput. Linguistics1, 2019, pp.4560–4565.

[17] C.Sugandhika and S.Ahangama, "Heuristics-Based SQL Query Generation Engine," 2021 6th International Conference on Information Technology Research (ICITR), Moratuwa, Sri Lanka, 2021, pp. 1-7, doi: 10.1109/ICITR54349.2021.9657317.

[18] A. Anisyah, T. E. Widagdo and F. Nur Azizah, "Natural Language Interface to Database (NLIDB) for Decision Support Queries," 2019 International Conference on Data and Software Engineering (ICoDSE), Pontianak, Indonesia, 2019, pp. 1-6, doi: 10.1109/ICoDSE48700.2019.9092769.

[19] X. Zhang, F. Yin, G. Ma, B. Ge and W. Xiao, "M-SQL: Multi-Task Representation Learning for Single-Table Text2sql Generation," in IEEE Access, vol. 8, pp. 43156-43167, 2020, doi: 10.1109/ACCESS.2020.2977613

[20] I.AndroutsopoulosI, G.Ritchie & P.Thanisch "Masque/sql-An Efficient and Portable Natural Language Query Interface for Relational Databases".

[21] Muhammad Shahzaib Baiget, Azhar Imran, Aman Ullah Yasin, Abdul Haleem Butt & Muhammad Imran Khan "Natural Language to SQL Queries: A Review", International Journal of Innovations in Science and Technology, 2022, vol. 4, pp.147-162, doi:10.33411/IJIST/2022040111.

[22] B.Sujatha, S.Vishwanadha Raju "Natural Language Query Processing for Relational Database using EFFCN Algorithm",International Journal of Computer Sciences and Engineering, 2016, vol.4.

[23] Abhishek Kharade "Generation of SQL Query Using Natural Language Processing",2023.

[24] Adrián Bazaga, Nupur Gunwant & Gos Micklem ""Translating synthetic natural language to database queries with a polyglot deep learning framework", 2021, vol.11, pp.18462, doi:10.1038/s41598-021-98019-3.

[25] Niculae Stratica, Leila Kosseim and Bipin C. Desai "NLIDB Templates for Semantic Parsing", 8th International Conference on Applications of Natural Language to Information Systems, Germany, 2003, pp.235-241

[26] T.H.Y. Vuong, T.T.T. Nguyen, N.T. Tran, L.M. Nguyen and X.H.Phan, "Learning to Transform Vietnamese Natural Language Queries into SQL Commands," 2019 11th International Conference on Knowledge and Systems Engineering (KSE), Vietnam, 2019, pp. 1-5, doi: 10.1109/KSE.2019.8919393.

[27] Gauri Rao, Chanchal Agarwal, Snehal Chaudhry, Nikita Kulkarni and S.H. Patil "Natural Language Query Processing using Semantic Grammar", International Journal on Computer Science and Engineering, 2010, vol.2, pp. 219-223.

[28] P. Pasupat and P. Liang, "Compositional semantic parsing on semi-Structured tables", 2015, pp. 1470–1480, doi:10.3115/v1/P15-1142.

[29] Amit Pagrut, Shambhoo Kariya, Vibhavari Kamble and Ishant Pakmode," Automated SQL Query Generator by Understanding a Natural Language", International Journal on Natural Language Computing, vol.7, pp.01-11, doi:10.5121/ijnlc.2018.7301.

[30] R. Kumar and M. Dua, "Translating controlled natural language query into SQL query using pattern matching technique," International Conference for Convergence for Technology-2014, Pune, India, 2014, pp. 1-5, doi: 10.1109/I2CT.2014.7092161.