SQL Query Generation using Transformers

Dr. David Raj Micheal
Division of Mathematics
School of Advanced Sciences
Vellore Institute of Technology Chennai
Tamil Nadu – 600127
davidraj.micheal@vit.ac.in

Anusha Puppala
Division of Mathematics
School of Advanced Sciences
Vellore Institute of Technology Chennai
Tamil Nadu – 600127
puppala.anusha2023@vitstudent.ac.in

***Abstract*—This project implements an SQL query generator based on the Transformers library, utilizing a large language model to automate SQL query generation from natural language questions. By leveraging Hugging Face's `sqlcoder-7b-2` model and GPU acceleration, the tool interprets user queries, assesses the provided database schema, and outputs optimized SQL statements. This approach aims to support non-technical users in interacting with relational databases without requiring expertise in SQL.**

I. Introduction

Relational databases are widely used in various domains for managing structured data. SQL (Structured Query Language) is essential for querying and managing data within these systems. However, writing SQL queries can be challenging for individuals without technical backgrounds. Recent advancements in Natural Language Processing (NLP) provide solutions where language models can translate natural language instructions into SQL queries. This project implements an automated SQL generation model using Hugging Face's `sqlcoder-7b-2` model, aiming to reduce the learning curve for users interacting with databases.

II. Problem Statement

The ability to transform questions into SQL queries can democratize data access within organizations, but achieving high accuracy and relevance in generated queries remains challenging. The project's primary objective is to:

1) Develop an NLP-based solution that converts natural language questions into SQL queries.
2) Ensure that generated queries are both efficient and compatible with the database schema.

III. Methodology

A. Imports and Model Setup

Key libraries and tools used include:

- `torch`: Manages tensor operations and GPU-based model loading.
- `transformers`: Loads Hugging Face's `sqlcoder-7b-2` model, providing the language capabilities for SQL generation.
- `bitsandbytes`: Allows model quantization to save memory on lower-spec devices.

The GPU memory availability is checked using `torch.cuda.get_device_properties()`. Based on the available memory, the model is loaded in either `float16` precision or `8-bit` mode. This selection helps manage performance and memory efficiency.

B. Prompt Creation

The model is guided by a structured prompt that combines:

- **Task description**: An instruction to generate an SQL query.
- **Additional rules**: A set of logical instructions defining common calculation rules.
- **Database schema**: A detailed schema of tables and relationships, allowing the model to accurately interpret the user's query in context.

The prompt template is structured to include placeholders, which are later replaced by specific questions.

C. Function to Generate SQL Queries

The function `generate_query(question)` integrates several steps:

- **Prompt Update**: The base prompt is updated with the user's specific question.
- **Tokenization**: `AutoTokenizer` processes the prompt, converting it into tensors.
- **Model Prediction**:

$$\text{Generated\_ids} = \text{model.generate}(inputs, \text{num\_return\_sequences} = 1 \tag{1}$$

Here, the model generates SQL statements based on tokens, stopping at defined sentence-ending tokens (e.g., `eos_token_id`).

- **Decoding and Formatting**: `sqlparse` cleans up the raw SQL output, formatting it to a readable structure.

To ensure memory efficiency, `torch.cuda.empty_cache()` clears any residual GPU memory, which is critical in environments like Google Colab.

D. Usage

Two example questions demonstrate the SQL generation process:

- **Question 1**: ”What was the highest quantity sold last month?”
- **Question 2**: ”Which salesperson sold a large amount of products last month?”

E. Example Output

Given the database schema, an example SQL query might look as follows: [language=SQL] SELECT ”name”, SUM(”price” * ”quantity”) AS total$_{r}evenue, SUM(''supply_{p}rice'' * ''quantity'')AStotal_{c}ostFROMproductsGROUPBY''name''$;

IV. Result

Upon testing, the SQL generation model successfully interpreted natural language questions into accurate SQL statements, confirming the ability of the `sqlcoder-7b-2` model to handle typical database operations, such as aggregation and join-based queries. The outputs were validated against the schema to ensure alignment with database constraints and structures.

V. Conclusion

This project demonstrates that language models like `sqlcoder-7b-2` can translate natural language questions into SQL queries efficiently and accurately, enabling non-technical users to interact with databases intuitively. While effective for basic and intermediate SQL queries, future work may focus on refining model responses for more complex queries and optimizing the prompt to improve understanding of intricate database relationships. Integrating error-checking mechanisms would further enhance reliability in practical applications.