2023- 2024

NLP PROJECT Report

TEXT-TO-SQL

Members :

- **Messaoudi Nabehet**
- **Meribai Khouloud**
- **Khodja Yousra**
- **Bouazza Ayat**

Prepared For

Dr R.Bousmaha

TABLE OF CONTENT

Abstract

This project focuses on developing a system called "Text to SQL" that translates natural language questions into SQL queries. The goal is to enable users, regardless of their technical background, to interact with databases using everyday language. By leveraging natural language processing (NLP) techniques, the system parses and interprets user queries, identifying key entities and relationships to construct accurate SQL queries. Our approach includes a deep learning model for semantic parsing, incorporating attention mechanisms and encoder-decoder architectures to handle varying sentence structures and database schemas. The system's performance is evaluated on its ability to accurately translate a diverse range of queries into SQL, ensuring robustness and usability. This work contributes to bridging the gap between non-technical users and database querying, making data-driven decision-making more accessible and intuitive.

Introduction

- **Context :**

Automating database querying is a critical challenge. Databases contain a massive amount of information that is essential for many businesses and organizations. However, querying these databases can be tedious and often requires extensive knowledge of the SQL language. This creates a barrier for non-technical users who could benefit from having access to this data. Thus, automating this process via NLP models has significant potential to make querying databases more accessible and efficient.

- **Motivation :**

Develop an NLP model to translate questions into SQL queries. This model will allow users to formulate questions in natural language and automatically convert them into SQL queries, making it easier to access data stored in relational databases without requiring extensive SQL knowledge.

- **Objectif :**

Apply NLP methods learned in the semester while exploring new approaches to improve the accuracy and robustness of the model. The goal is to leverage recent advances in NLP to create a system that can understand a wide range of questions in natural language and accurately translate them into SQL queries.

Chapter 1 : Methodology

- ## Dataset Description :

The dataset used in this study has three columns: Context: The context in which the question was asked and the answer was given (in text format). Question: The question asked. SQL: The corresponding SQL query. The dataset contains a single file, titled train.csv. It is designed to help researchers better understand how factors such as semantics and context influence interpretation and response to different conversations on various topic .Here is the link to the dataset:

https://www.kaggle.com/datasets/thedevastator/understanding-contextual-questions-answers/data

- ## Dataset Exploration :

context The context in which the question was asked and the answer was given. (Text)	question Question	answer Answer
72947 unique values	**78311** unique values	**78577** unique values
CREATE TABLE head (age INTEGER)	How many heads of the departments are older than 56 ?	SELECT COUNT(*) FROM head WHERE age > 56
CREATE TABLE head (name VARCHAR, born_state VARCHAR, age VARCHAR)	List the name, born state and age of the heads of departments ordered by age.	SELECT name, born_state, age FROM head ORDER BY age

Summary

1 file

3 columns

- # Models Implemented :

a. Gemma Model

Gemma Model is a collection of lightweight, open-source generative AI (GenAI) models developed by Google DeepMind. These models are primarily aimed at developers and researchers. Gemma was launched alongside Gemini, Google's closed-source generative AI chatbots. There are two main models in the Gemma collection: Gemma 2B and Gemma 7B. These models are text-to-text decoder grand language (LLM) models with pre-trained and instruction-tuned variants. Gemma 2B has a neural network with 2 billion parameters, while Gemma 7B has a neural network with seven billion parameters. Google offers pre-trained, instruction-tuned Gemma models suitable for running on laptops and workstations. These templates are available to developers through various platforms. Additionally, Meta's Llama 2 is another open-source AI model designed to run on laptops...

b. T5 (Text-To-Text Transfer Transformer)

T5 is a natural language processing model developed by Google Research. Introduced in the article "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer", T5 takes a unified approach where all NLP tasks, such as translation, summarizing, classification, and response generation, are framed as text-to-text conversion problems. This approach allows the model to use a transform architecture to encode input text sequences and generate output text sequences, making it easier to handle a wide variety of linguistic tasks with a single model.

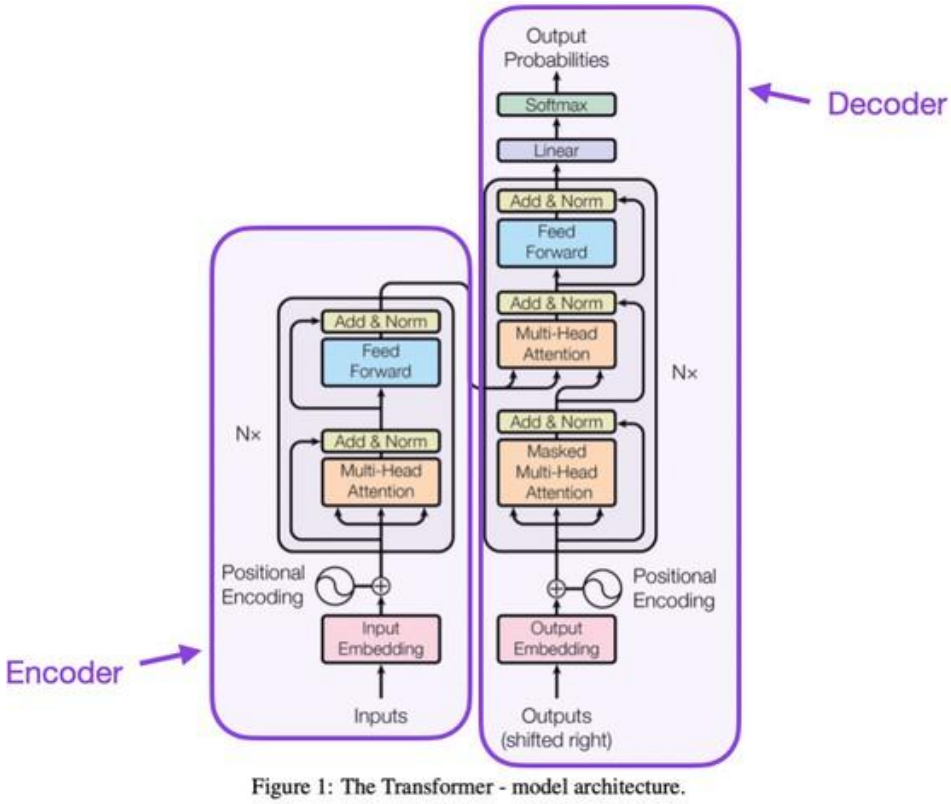
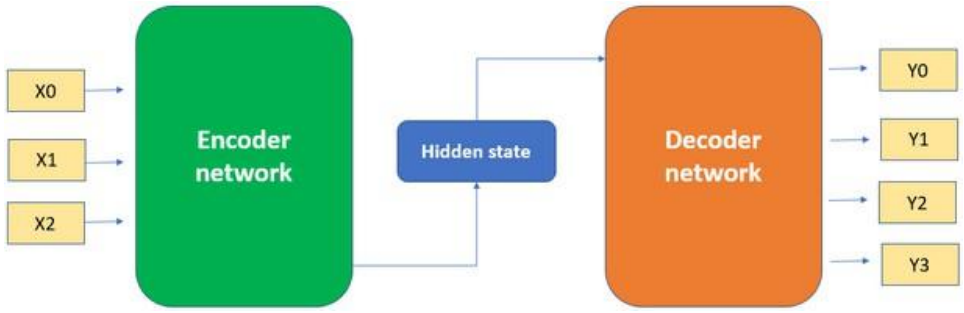


Figure 1: The Transformer - model architecture.

c. Encoder-Decoder (from scratch)

The Encoder-Decoder architecture is a fundamental model in natural language processing (NLP) that consists of two main parts: the encoder and the decoder. The encoder takes an input sequence (e.g., a sentence) and transforms it into a compressed contextual representation, capturing the essential information and relationships of the sequence. The decoder then uses this representation to generate an output sequence (e.g., a translation) word by word. This architecture is widely used for tasks such as machine translation, text summarization, and text generation, thanks to its ability to efficiently process sequences and produce accurate results by capturing complex contextual dependencies.



- # Evaluation Metrics :

1. BLEU Score (Bilingual Evaluation Understudy)

·**Description:**

The BLEU score is a metric for evaluating the quality of text generated by a model compared to one or more reference texts. It is widely used in machine translation and text generation tasks. BLEU score measures how many n-grams (up to four) in the generated text appear in the reference text, with a penalty for shorter generated texts.

·**Calculation:**

Ø Tokenize the predicted SQL query and the reference SQL query.

Ø Calculate n-gram precision for n = 1 to 4.

Ø Apply a brevity penalty to discourage short translations.

Ø Combine the n-gram precisions into a single score.

2. WRE (Word Retrieval Error)

·Description:

Word Retrieval Error (WRE) measures the proportion of incorrect words in the predicted SQL queries compared to the reference queries. It helps in identifying the accuracy of word-level predictions in the SQL output.

·Calculation:

Ø Count the number of words in the predicted query that do not match the reference query.

Ø Divide by the total number of words in the reference query.

3. Precision

·Description:

Precision measures the proportion of correctly predicted positive observations to the total predicted positives. It is particularly useful for evaluating the accuracy of the SQL tokens that the model predicts.

·Calculation:

Ø True Positives (TP): Correctly predicted tokens.

Ø False Positives (FP): Incorrectly predicted tokens.

4. Recall

·Description:

Recall measures the proportion of correctly predicted positive observations to the all observations in actual class. It helps in assessing how well the model retrieves relevant tokens.

·Calculation:

Ø True Positives (TP): Correctly predicted tokens.

Ø False Negatives (FN): Relevant tokens that were not predicted.

5. F-measure (F1 Score)

·Description:

The F-measure, or F1 score, is the harmonic mean of Precision and Recall. It provides a single metric that balances both concerns, making it a comprehensive measure of the model's performance.

Chapter 2 : Implementation

a. Gemma Model :

1. **Preprocessing**

The dataset is prepared for fine-tuning by iterating over each row in the training data and extracting the 'question' and 'answer' fields. For each pair, a formatted template is created, placing the question and answer into a structured text format with clear labels and line breaks for readability. This template is then appended to a list named dataset. The resulting list contains all the formatted question-answer pairs, making it ready for fine-tuning a language model to generate answers from questions based on the provided examples**.**

2. **Training**

- **Fine tuning with LoRA:**

LoRA (Low-Rank Adaptation of Large Language Models) is a technique used to improve the efficiency and performance of large language models by significantly reducing the number of training parameters. The line enables LoRA specifically on the backbone architecture of the language model gemma_lm**.**

- **Memory Control – Epochs:**

Compiling the model using the specified loss function, optimizer, and metrics

o SparseCategoricalCrossentropy: It is used as a loss function to calculate the cross-entropy loss between the true labels and the predicted probabilities generated by the model. It measures the difference between the actual distribution of the labels and the planned distribution.

o Adam optimizer: Effectively optimizes the parameters (weight and bias) of the neural network during training. It adapts the learning rate for each parameter individually based on the estimates of the first and second gradient moments. SparseCategoricalAccuracy is used as an evaluation metric, which calculates accuracy during training.

3. **Results**

The developed model performed well, demonstrating its ability to efficiently capture the structure of SQL queries. Through training on the Create Context SQL dataset, the model learned how to translate natural language questions into precisely structured SQL queries. The results obtained show that the model manages to generate correct and consistent SQL queries, indicating a good understanding of the relationships between the elements of the question and the SQL syntax required to obtain the desired information.

Ø Some examples of prompting :

```
Question: What are the maximum and minimum budget of the departments

Answer:

SELECT MAX(budget), MIN(budget) FROM departments
```

```
Question:
What are the names of the heads who are born outside the California state?

Answer:
SELECT name FROM heads WHERE birth_location = "outside California"
```

b. Encoder_Decoder:

1. **Preprocessing**

• 1. Removing Question Marks from the Question Sentences

the first preprocessing step is to clean the question sentences by removing any question marks ('?'). This ensures that the model is not influenced by the punctuation, which does not contribute to the actual meaning of the question.

2. Converting All Questions and SQL Queries to Lower Case: to maintain consistency and reduce variability in the data.

3. Tokenizing Both Question and SQL Queries

Tokenization is the process of breaking down the text into individual tokens, which is crucial for converting the text into a format suitable for further processing and model training.

4. Encoding the Question and SQL Queries into Sequences of Indices

After tokenization, each token is mapped to a unique index using a vocabulary dictionary. This transforms the tokens into sequences of numerical indices.

5. Padding the Sequences to Have the Same Length

To handle batches of data efficiently, all sequences are padded to the same length. Padding ensures that all sequences in a batch have uniform length, which is necessary for matrix operations in deep learning models. Padding is usually done by adding a special token (e.g., 0) at the end of shorter sequences.

6. Splitting Data into Train, Validation, and Test Sets

Finally, the data is split into three sets: training, validation, and test sets.

Model 1: Simple Bidirectional LSTM Encoder-Decoder

Description:

This model is a straightforward encoder-decoder architecture with Bidirectional LSTM layers. The encoder processes the input sequence, and the RepeatVector layer repeats the encoder's output to match the target sequence length. The decoder, another Bidirectional LSTM, generates the output sequence. The final layer is a TimeDistributed Dense layer with a softmax activation for outputting a probability distribution over the target vocabulary at each timestep.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 35, 512)           23307776

 bidirectional (Bidirection  (None, 1024)              4198400
 al)

 repeat_vector (RepeatVecto  (None, 30, 1024)          0
 r)

 bidirectional_1 (Bidirecti  (None, 30, 1024)          6295552
 onal)

 time_distributed (TimeDist  (None, 30, 42123)         43176075
 ributed)

=================================================================
Total params: 76977803 (293.65 MB)
Trainable params: 76977803 (293.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model 2: Bidirectional LSTM Encoder-Decoder with Dropout and Additional Dense Layer

Description:

This model enhances the basic Bidirectional LSTM encoder-decoder structure by incorporating dropout for regularization, which helps prevent overfitting. It also includes an additional TimeDistributed Dense layer with ReLU activation before the final softmax layer, potentially allowing the model to learn more complex representations.

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 30, 256)           2560000

 bidirectional_17 (Bidirect  (None, 512)               1050624
 ional)

 repeat_vector (RepeatVecto  (None, 35, 512)           0
 r)

 bidirectional_18 (Bidirect  (None, 35, 512)           1574912
 ional)

 time_distributed_13 (TimeD  (None, 35, 10000)         5130000
 istributed)

=================================================================
Total params: 10315536 (39.35 MB)
Trainable params: 10315536 (39.35 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model 3: Bidirectional LSTM Encoder-Decoder with Attention Mechanism

Description:

This model employs a sophisticated encoder-decoder structure with multi-layer Bidirectional LSTM units and an attention mechanism. The encoder consists of two Bidirectional LSTM layers, whose states are concatenated to form the initial state for the decoder. The decoder uses a unidirectional LSTM with an attention layer that aligns the decoder's input with relevant encoder outputs, enhancing the model's ability to handle long sequences and complex dependencies. The final layer is a Dense layer with softmax activation to generate the output sequence.

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 46)]              0         []

embedding (Embedding)        (None, 46, 256)           10147328  ['input_1[0][0]']

bidirectional (Bidirectional [(None, 46, 512),         1050624   ['embedding[0][0]']
)                             (None, 256),
                              (None, 256),
                              (None, 256),
                              (None, 256)]

input_2 (InputLayer)         [(None, 115)]             0         []

bidirectional_1 (Bidirectional [(None, 46, 512),       1574912   ['bidirectional[0][0]']
)                             (None, 256),
                              (None, 256),
                              (None, 256),
                              (None, 256)]

embedding_1 (Embedding)      (None, 115, 256)          10383104  ['input_2[0][0]']

concatenate (Concatenate)    (None, 512)               0         ['bidirectional_1[0][1]',
                                                                  'bidirectional_1[0][3]']

concatenate_1 (Concatenate   (None, 512)               0         ['bidirectional_1[0][2]',
)                                                                 'bidirectional_1[0][4]']

lstm_2 (LSTM)                [(None, 115, 512),        1574912   ['embedding_1[0][0]',
                              (None, 512),                        'concatenate[0][0]',
                              (None, 512)]                        'concatenate_1[0][0]']

attention (Attention)        (None, 115, 512)          0         ['lstm_2[0][0]',
                                                                  'bidirectional_1[0][0]']

concatenate_2 (Concatenate   (None, 115, 1024)         0         ['attention[0][0]',
)                                                                 'lstm_2[0][0]']

dense (Dense)                (None, 115, 40559)        41572975  ['concatenate_2[0][0]']

==================================================================================================
Total params: 66303855 (252.93 MB)
Trainable params: 66303855 (252.93 MB)
Non-trainable params: 0 (0.00 Byte)
```

2. **Training**

All three models are trained using the Adam optimizer and sparse categorical cross-entropy loss. The training process monitors the loss to determine when to stop training early if the validation loss does not improve, preventing overfitting and ensuring the best model is saved.

	Trainig_loss	Val_loss	Traning_duration	Bleu score	wre	presion	recall	Fmesure	epochs
Model1	0.35	1.35	30H	0.24	0.66	0.71	0.44	0.54	40
Model2	1.6	2.8	10H	0.26	0.64	0.72	0.44	0.54	10
Model3	0.35	0.79	5H	0.06	0.81	0.59	0.35	0.44	30

3. **Results**

Model 1 exhibits overfitting, with decent precision but lower recall, indicating that it struggles to retrieve all relevant tokens.
Model 2 shows some improvements in generating correct sequences and word retrieval compared to Model 1 but suffers from higher losses, likely due to insufficient training epochs.
Model 3 generalizes better with lower validation loss but has poor sequence generation quality and high word retrieval errors, despite the attention mechanism. It may require further tuning.

c. Simple Seq2Seq Model :

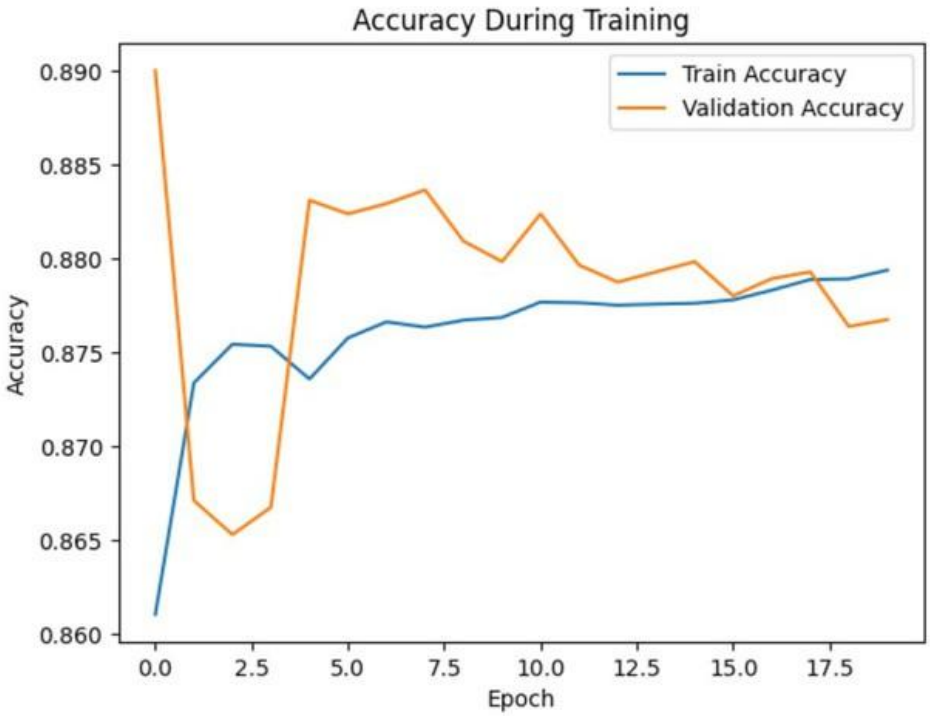
We decided to try a more simple approach by creating a simple Seq2Seq architecture with LSTM layer**s.**

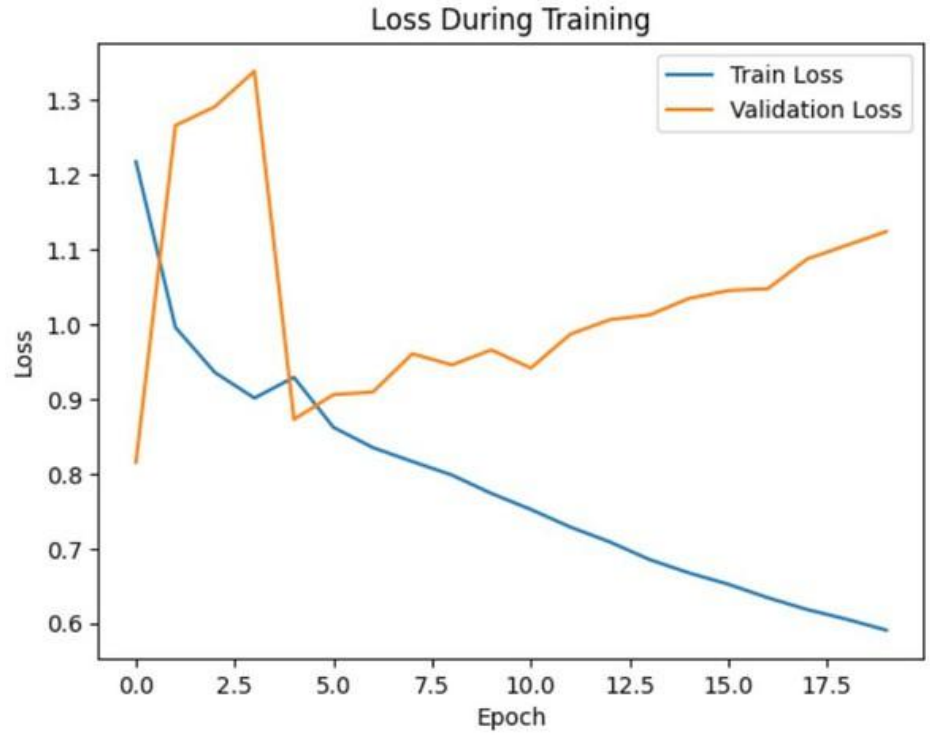
1. **Preprocessing**

To prepare the data, we started by shuffeling it to ensure randomness then split it into training, testing, and validation sets based on specified sizes. We started the preprocessing by defining two simple preprocessing functions that clean and standardize the questions and SQL queries, involving steps like converting text to lowercase, removing punctuation, extra whitespace, and stopwords. The dataset is then tokenized and converted into sequences using TensorFlow's tokenizer, which transforms the text into numerical data suitable for model input. The sequences are padded to ensure uniform input length, accommodating the requirements of the LSTM layers in the Seq2Seq model. This comprehensive preprocessing pipeline ensures the data is in the optimal format for subsequent model training and evaluation.

2. **Training**

The model includes an LSTM-based encoder-decoder architecture where the encoder processes input sequences, and the decoder generates the output sequences. The encoder's final hidden state is repeated and fed into the decoder, which outputs sequences with softmax activation to predict vocabulary probabilities. The model is then compiled with sparse categorical crossentropy loss and the Adam optimizer, and accuracy as the metric. During training, the model is trained over 20 epochs utilizing training and validation data.





Although the accuracy results are promising, the loss graph shows significant signs of overfitting. The test loss of 1.199 and test accuracy of 0.871 indicate that the model performs reasonably well on unseen data, maintaining a high accuracy. However, these results suggest that accuracy may not be the most appropriate metric for this task. Therefore, we will explore additional evaluation metrics in the following section to obtain a more comprehensive assessment of the model's performance.

3. **Results**

The evaluation was conducted using various metrics such as BLEU score, ROUGE score, structural similarity, and visual inspection of generated queries against the ground truth.
Firstly, BLEU score and ROUGE score metrics are employed to quantitatively assess the quality of the model's predictions. BLEU score measures the similarity between the predicted and reference answers based on n-gram overlap, while ROUGE score evaluates the overlap in n-gram sequences and the longest common subsequence between the predicted and reference answers. The model achieves an average BLEU score of 0.60 and average ROUGE scores of 0.45 (ROUGE-1), 0.15 (ROUGE-2), and 0.43 (ROUGE-L), indicating moderate performance in capturing the semantics and structure of the reference answers.
Next, the structural similarity metric assesses the syntactic similarity between the predicted and reference SQL queries by comparing their parsed representations. The average structural similarity achieved by the model is 44.00%, suggesting that the model struggles with replicating the structural nuances of the reference queries accurately. Calculating the average similarity between the model's predicted SQL queries and the reference queries in the test dataset offers a quantitative insight into the model's effectiveness in generating accurate SQL representations from the input questions. In this case, the average similarity is 57.94%, which indicates that, on average, the generated SQL queries to a degree resemble the reference queries in structure and content.
Additionally, qualitative evaluation is performed by visually inspecting a subset of generated SQL queries alongside their corresponding ground truth queries. This inspection provides insights into specific instances where the model succeeds or fails in generating accurate SQL queries based on the input questions. We notice that the result are not as good as the previous metrics indicated which leads us to conclude that visual inspecting is the better evaluation metric for our task.

d. T5 Model :

1. **Preprocessing**

We began by loading a dataset from a CSV file and created a subset of **2000** samples for training, validation, and testing using train_test_split. The data was split into 60% for training, 20% for validation, and 20% for testing. We then initialized the T5 tokenizer and model from the Hugging Face Transformers library. Using the tokenize_data function, we tokenized the questions and corresponding SQL queries. Each question was prefixed with "translate English to SQL: " and both inputs and targets were encoded, padded, and truncated to a maximum length of 512 tokens. The tokenized data was converted into PyTorch TensorDataset objects, and DataLoader objects were created for each dataset with a batch size of 1.for the model, and special tokens are used for padding and truncation to ensure consistent input sizes.

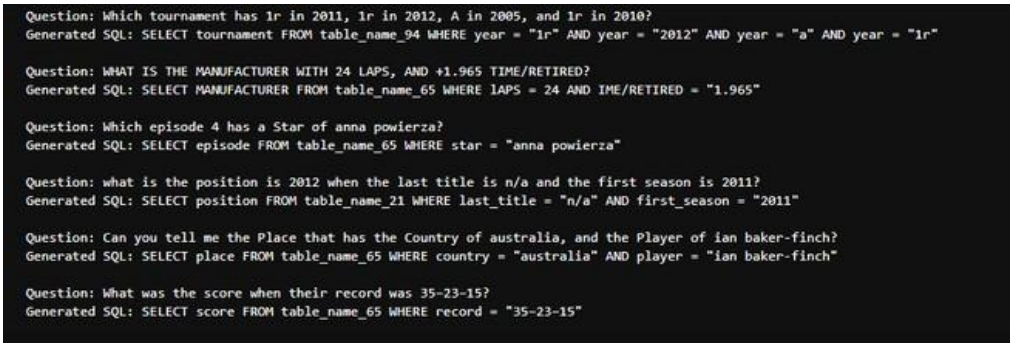
2. **Training**

For training, we used the **AdamW** optimizer with a learning rate of **5e-5** and set up a linear learning rate scheduler with warmup steps. The **train_model** function handled the training and validation process. This function iterated over epochs and training batches, computed model outputs and loss, and applied gradient accumulation for parameter updates. Predictions and true labels were collected to calculate accuracy, and average loss and accuracy were printed per epoch. After each epoch, the model was validated on the validation dataset, with validation loss and accuracy computed and printed. The training process was executed for **10** epochs with **16** accumulation steps, ensuring efficient training and performance monitoring through validation.

3. **Results**

The model was trained for 10 epochs with a gradient accumulation step of 16. The average training loss by the end of the training process was 1.0665. The validation results showed an average loss of 1.0021, indicating that the model's performance on the validation set was slightly better than on the training set. In addition to loss, we evaluated the model using BLEU and ROUGE scores, which are common metrics for natural language generation tasks. The BLEU score was 0.4459, reflecting the precision of the generated SQL queries compared to the reference queries. The ROUGE scores were as follows: ROUGE-1 (measuring the overlap of unigrams) was 0.7470, ROUGE-2 (measuring the overlap of bigrams) was 0.5077, and ROUGE-L (measuring the longest common subsequence) was 0.7272. These scores suggest that the model was reasonably effective at generating SQL queries that have substantial overlap with the reference queries in terms of both unigrams and bigrams, as well as in the longest common subsequence.
The total training time was approximately 6 hours. Despite the challenges reflected in the training and validation losses, the BLEU and ROUGE scores indicate that the model can generate SQL queries with a fair degree of similarity to the target queries, which is promising for further fine-tuning and optimization.

Here are some of the predictions :



Discussion :

Challenges:

1. Training Time and Resource Constraints:
   - Long Training Times: Training advanced NLP models such as BERT or GPT for SQL query generation is time-consuming due to the complexity of the models and the large amount of data required for effective training.
   - Resource Availability: Limited access to high RAM and CPU power can hinder the ability to train and deploy these models effectively, posing a significant challenge for organizations with constrained computational resources.

2. Complexity of Questions:
   - Diverse Query Types: Natural language queries for SQL generation vary widely in complexity and structure, ranging from simple retrieval tasks to complex joins, aggregations, and nested queries.
   - Ambiguity and Precision: Ambiguities in natural language can lead to misunderstandings, making it challenging for models to accurately discern user intent and translate it into precise SQL queries.

3. Accuracy Limitations:
   - Edge Cases: Handling uncommon query patterns or unexpected inputs can challenge the model's ability to accurately interpret and generate SQL queries.
   - Error Handling: Ensuring robust error handling mechanisms are in place to provide informative feedback when the model cannot process a query accurately is crucial for user experience and system usability.

In summary, developing models for generating SQL queries from natural language faces significant challenges related to training time, resource constraints, and the complexity of natural language queries. These challenges need to be carefully addressed to build accurate and reliable systems.

Chapter 3 : Deployment

In order to democratize database querying and make it accessible to users without technical SQL knowledge, we have deployed the "Text to SQL" model in Streamlit. This integration transforms the interaction between natural language questions and SQL databases into a user-friendly experience. Streamlit's intuitive web application interface allows users to input queries in everyday language, instantly generating SQL commands. By leveraging interactive widgets and real-time visualizations, we provide immediate feedback on query interpretation and results, empowering users to make data-driven decisions effortlessly. This deployment not only simplifies database querying but also enhances usability, enabling seamless interaction with complex data structures through straightforward natural language input.

Conclusion

In developing a system to convert natural language text into SQL queries, significant challenges such as query complexity, resource constraints, and accuracy limitations were encountered. Natural language queries vary greatly in complexity, requiring the system to handle diverse structures and ambiguities effectively. Limited access to high computational resources posed challenges in training and deploying advanced NLP models, impacting system performance. Addressing these challenges involved optimizing model architectures, implementing advanced query parsing techniques, and fine-tuning models on domain-specific data. Moving forward, ongoing research into more advanced NLP architectures and optimized resource management will be critical for further enhancing the system's accuracy and usability in generating SQL queries from natural language inputs.