DATS.6312 - Natural Language Processing

Final Project Report

Grou Report by

Bharath Genji Mohana Ranga
Mowzli Sre Mohan Dass

1. Abstract

The project, "Automating Text-to-SQL Conversion with Contextual Understanding," aims to bridge the gap between natural language and SQL query generation, addressing the challenges faced by non-technical users in database interactions. Leveraging advanced NLP techniques, the system integrates BERT for label prediction and T5 for SQL generation to translate user queries into precise SQL statements. Synthetic datasets generated through Llama3:8b models supplemented by diverse schema structures were used for training. The system employs preprocessing techniques like tokenization and similarity mapping, alongside beam search decoding for enhanced SQL accuracy. Evaluation metrics such as BLEU scores and classification precision indicate high accuracy, while regularization and data augmentation mitigate overfitting. Despite its success in handling ambiguity and schema diversity, the project identifies areas for improvement, including real-time adaptability and user interface enhancements. This work contributes to democratizing database access, enabling intuitive querying for users without SQL expertise.

2. Introduction

Problem Statement

Accessing databases is essential for decision-making in various fields. However, non-technical users often face barriers in writing SQL queries. This project focuses on developing a system that translates natural language queries into SQL to address this challenge.

Objectives

- Develop a robust natural language to SQL translation system.
- Address natural language ambiguity.
- Ensure the system is scalable and adaptable to diverse schema structures.
- Enhance user accessibility to databases by eliminating the need for SQL expertise.
- Create a versatile system that can adapt to various database schemas and query requirements.
- Improve the accuracy of SQL query generation using advanced NLP models.
- Build a modular pipeline that allows for easy integration with existing database systems.

- Provide detailed insights into model performance through metrics and explainability techniques.

3. About the Dataset

Initial Sources

Initially, we considered datasets like KaggleDBQA, Spider, SEDE, and SQL-Eval due to their richness in authentic sources, database-validated syntax, and human evaluation. However, given the project timeline, the dataset structures presented challenges in alignment with our project expectations. The syntaxes were in JSON format, and serializing the entire structure would have required significant changes to our pipeline, shifting the focus of the project toward data engineering. To address this, we opted to generate synthetic data tailored to our needs, enabling us to train the models effectively while staying aligned with our objectives.

Synthetic Data Generation

To generate a synthetic dataset, we opted to use the open-source Llama3 model from Meta, implemented through the Ollama pipeline. This approach significantly reduced the time and resources required for data collection. By leveraging proper prompt engineering, validation functions from the SQLite3 library in Python, and preprocessing techniques using Pandas, we successfully generated a dataset comprising 60,000 unique rows with Phrase and relevant SQL. This robust dataset provided a solid foundation for fine-tuning our chosen model, allowing us to effectively align the weights and meet the project's objectives.

4. NLP Model

Draft Pipeline

Initially, we planned to leverage the **T5ForConditionalGeneration model** for Text-to-SQL conversion, a robust choice for text-to-text translations. Developed by Google, the T5 model series where *"It's an encoder decoder transformer pre-trained in a text-to-text denoising generative setting."*. Our workflow integrated multiple NLP pipelines for dataset preprocessing, fine-tuning the model, addressing schema similarity, and predicting relevant SQL queries. However, we encountered challenges in conveying adequate knowledge to the model. Embedding this knowledge in the NLP preprocessing stage led to anomalies in the prompt, causing the model to misinterpret the correct intent.

To resolve this, we shifted the focus to the fine-tuning stage. Here, the model was trained using a combination of text prompts and SQL labels, enabling it to better streamline intent and avoid diversions. This refinement significantly improved the model's ability to generate precise SQL queries, aligning closely with our project objectives.

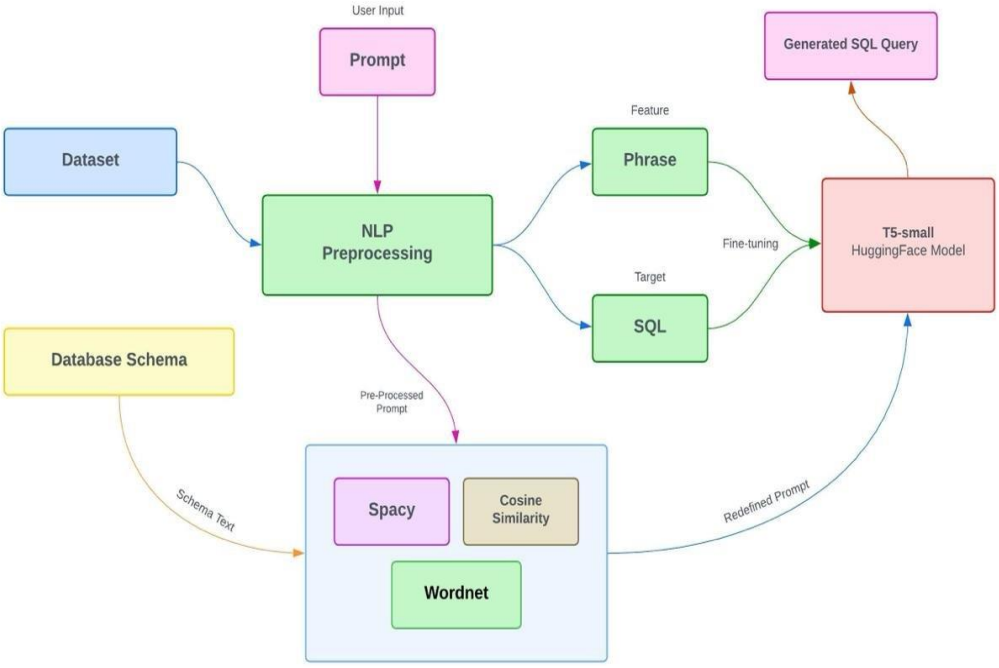

Fig. 1 Initial Workflow of Text-SQL using T5ForConditionalGeneration model

Final Approach

While working with intents, we evaluated various approaches such as vector embedding, transformer embedding, and classification labels. Vector embeddings, which represent intents as high-dimensional numerical vectors, could provide the necessary knowledge to the transformer model. However, in our case, the semantic knowledge was already incorporated during the redefinition of the prompt. Using vector embeddings specifically for SQL knowledge would be computationally expensive and introduce additional complexity to the pipeline.

Transformer embeddings, while specific to the model, reduced the granularity of meaning and introduced noise to the predictions, making them less suitable for our use case. Consequently, we opted for a Multilabel Classification BERT model pipeline, which identifies relevant SQL labels from the prompt and utilizes them for training in our custom T5 model.

This approach proved to be straightforward and efficient. We employed a simplified transformer, specifically the BERT base uncased model, to classify the labels. *"This is different from traditional recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally masks the future tokens. It allows the model to learn a bidirectional representation of the sentence."* This bidirectional capability of BERT enables it to identify relevant labels from tokens effectively.

Using the BertTokenizer, the prompt was tokenized and classified into various basic SQL syntaxes such as SELECT, WHERE, GROUP BY, HAVING, ORDER BY, ASC, DESC, LIMIT,

OFFSET, LIKE, BETWEEN, IN, IS NULL, and IS NOT NULL. For this project, we focused on training the model to handle basic SQL generation. Once the model demonstrates competency in this domain, it can be further trained to predict more complex SQL syntaxes. The classified labels, combined with the prompt, facilitate clear and accurate SQL predictions.

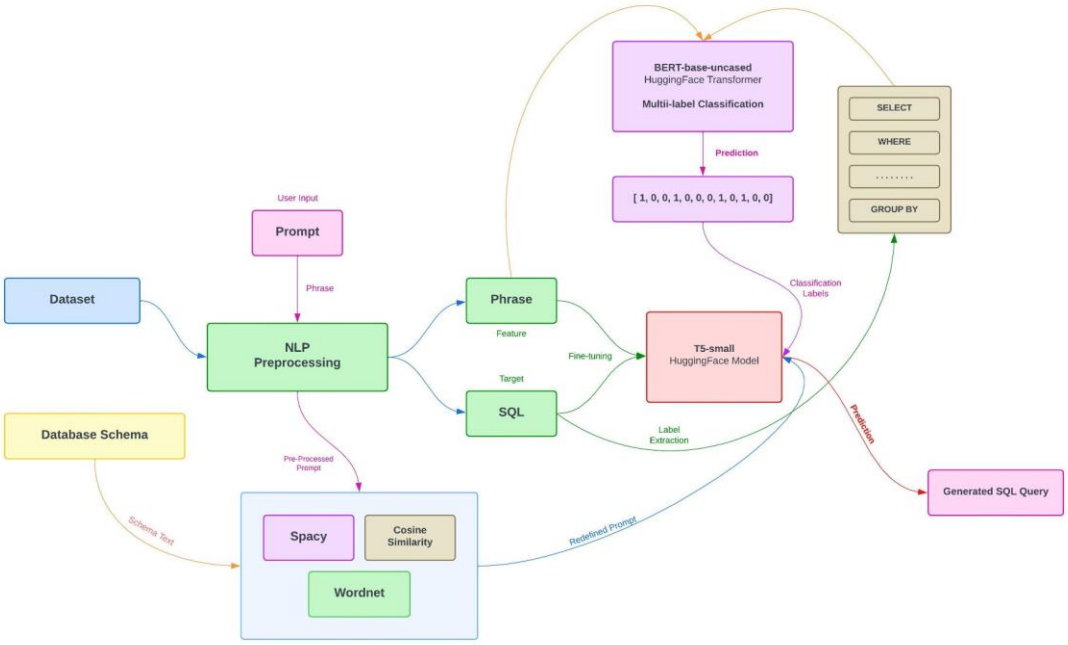

Fig. 2 Final Workflow of Text-SQL using BERT Multilabel Classification and T5 Prediction Model

5. Experimental Setup

Natural Language Preprocessing

During dataset preprocessing, we faced challenges in combining multiple techniques such as lemmatization, stemming, Bag of Words, Named Entity Recognition, and dependency parsing. Incorporating all these methods altered the nature of the prompts, resulting in non-concise phrases that hindered efficient predictions. To address this, we streamlined the preprocessing by focusing on essential techniques like stop-word removal, tokenization, and stemming. This approach produced clean and concise prompts, making them easier to interpret and translate effectively.

Similarity Mapping

The Similarity Mapping function plays a crucial role in our pipeline by redefining the user prompt to align with the database schema definitions. This step addresses the challenge of users providing prompts that do not exactly match the column names in the schema. While this issue could be avoided by requiring users to use precise schema terms, the Similarity Mapping feature enhances flexibility, allowing users to phrase their prompts freely while still ensuring accurate SQL syntax generation.

The pipeline dynamically generates synonyms for schema terms using WordNet, applies fuzzy matching for approximate term alignment, and validates the normalized prompt against the schema. By redefining the prompt with schema-relevant terms, this code enables users to query databases with flexibility, without requiring exact knowledge of schema terminology, while maintaining a high level of precision and relevance.

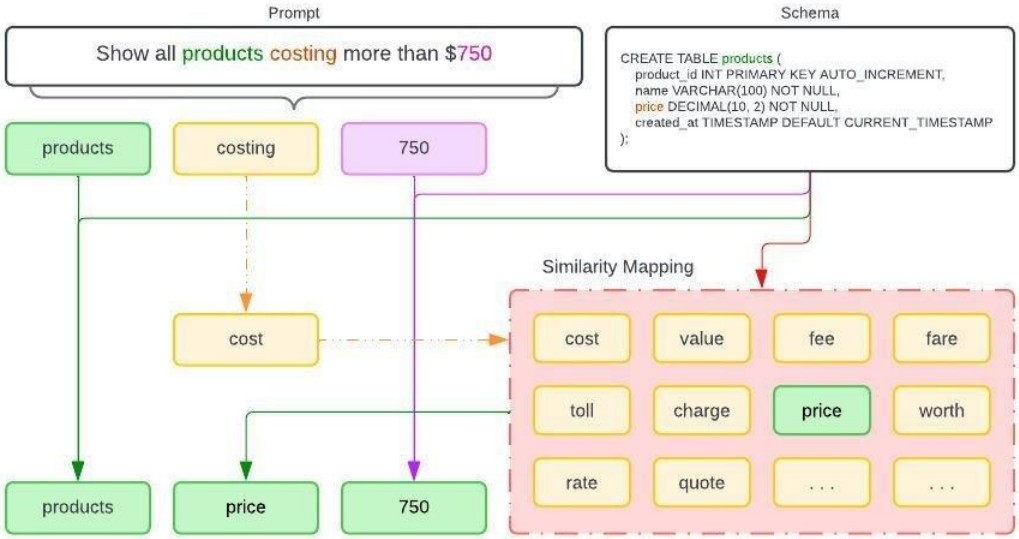

Fig. 3 Similarity Mapping of User prompt based on the Database Schema

Label Extraction

The label extraction process enhances the model's ability to generate accurate SQL queries by providing additional contextual knowledge. Key SQL components such as SELECT, WHERE, and GROUP BY are extracted from target SQL syntaxes and used during training to guide the model. These labels act as explicit markers that help the model understand the structure and purpose of each query, reducing ambiguity and aligning the output with the user's intent.

Incorporating labels into the training process makes learning more efficient and targeted. Rather than relying solely on natural language prompts, the model uses labels to prioritize relevant query components, improving accuracy and reducing errors. For instance, a WHERE label directs the model's focus to identifying filtering conditions, ensuring the generated SQL is precise even for complex queries with multiple clauses.

Phrase	SQL	SELECT	WHERE	GROUP BY	HAVING	ORDER BY	ASC	DESC	LIMIT	OFFSET	LIKE	BETWEEN	IN	IS NULL	IS NOT NULL
Show all cust	SELECT name	1	0	0	0	0	0	0	0	0	0	0	0	0	0
List all produ	SELECT categ	1	0	0	0	0	0	1	0	0	0	0	0	0	0
Display orde	SELECT orde	1	1	0	0	0	0	0	0	0	0	0	1	0	0
Get employe	SELECT name	1	1	0	0	0	0	0	0	0	0	0	1	0	0
Retrieve the	SELECT prod	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Show all cust	SELECT phon	1	0	0	0	0	0	0	0	0	0	0	0	0	0
List all order	SELECT orde	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Display the n	SELECT name	1	1	0	0	0	0	0	0	0	0	0	1	0	0
Get the emai	SELECT emai	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 4 Sample Dataset after Label Extraction

6. Model Setup

BERT Multilabel Classification Model

This implementation focuses on training a multi-label classification model using a custom architecture based on a pre-trained BERT model. The primary objective is to predict relevant SQL labels from text phrases, which enables accurate SQL generation in subsequent stages. The model combines BERT's contextual embeddings with convolutional and attention layers to capture both global and local semantic features of the input text, making it particularly suited for handling complex multi-label classification tasks.

The training pipeline begins with preprocessing the dataset. Text phrases are tokenized using BERT's tokenizer, which ensures compatibility with the pre-trained model's input format. The dataset is split into training and testing subsets, and a custom Dataset class is used to prepare batches for efficient loading and processing. Labels corresponding to SQL syntax components are encoded into multi-label arrays, providing the model with clear targets for training.

During training, the model leverages BERT's CLS token embedding as a global representation of the input sequence. It also uses convolutional layers to extract local features from token embeddings and an attention mechanism to identify the most relevant parts of the sequence. The outputs of these layers are combined and passed through fully connected layers to produce predictions for the SQL labels. A binary cross-entropy loss function is employed to handle the multi-label nature of the task, and the Adam optimizer fine-tunes the model parameters. Additionally, a learning rate scheduler helps optimize convergence during training.

To evaluate the model, metrics such as precision, recall, F1-score, and subset accuracy are computed. These metrics provide insights into how well the model captures the relationships between text phrases and SQL components. The use of evaluation metrics like Jaccard index and Hamming loss further highlights the model's ability to handle multi-label predictions effectively. After training, the model's weights and tokenizer are saved, enabling future inference tasks with consistent preprocessing and prediction workflows. This end-to-end training pipeline ensures the model is robust and ready for deployment in SQL label prediction tasks.

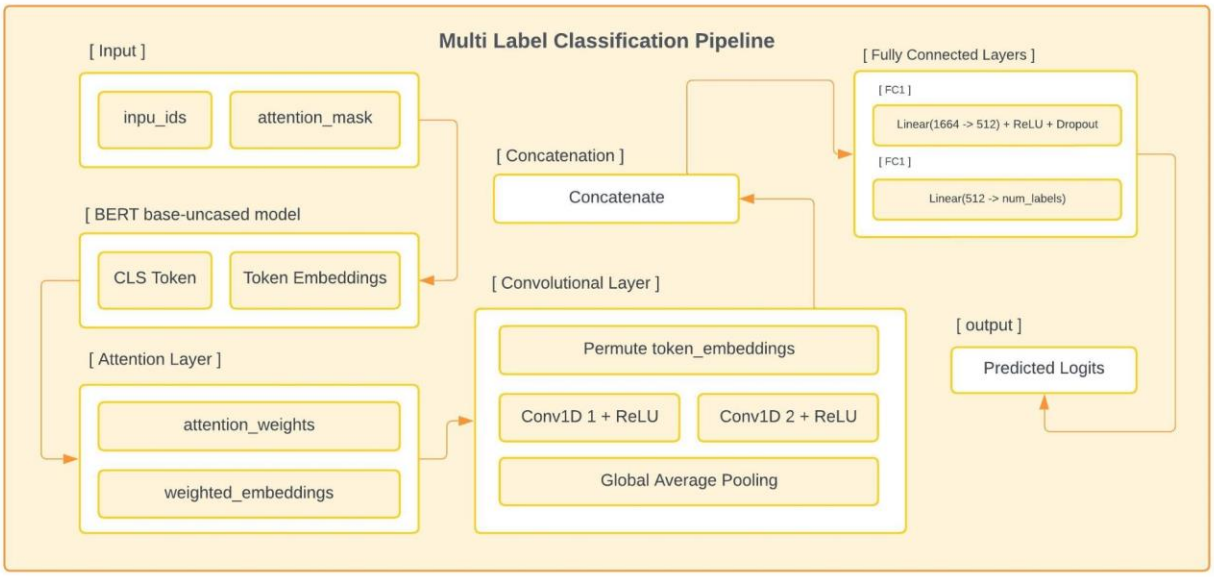

Fig. 5 Multilabel Classification Model using BERT-base-uncased

Custom T5ForConditionalGeneration SQL Prediction Model

The training pipeline begins with data preprocessing, where the input phrases are normalized by converting them to lowercase and stripping unnecessary spaces. The dataset contains natural language phrases, their corresponding SQL queries, and multi-label syntax components that provide context for the SQL query structure. A custom preprocessing function combines these elements into a unified input format, such as: *"SQL prediction: <phrase> [Labels: SELECT:1, WHERE:0]"*. This ensures the model receives both the natural language prompt and its associated SQL structure during training.

The Seq2SeqDataset class facilitates the preparation of training and validation data, handling tokenization, truncation, and padding using the T5 tokenizer. The T5 model takes these tokenized inputs, processes them through its encoder-decoder architecture, and generates SQL queries as output. The inclusion of labels in the input helps the model focus on the structural components of SQL syntax, reducing ambiguity and improving accuracy.

Training the model involves minimizing the loss between the generated and target SQL queries using the AdamW optimizer. The training process is enhanced by evaluating the model on validation data after each epoch, allowing for the monitoring of both training and validation loss. By iteratively fine-tuning the model over several epochs, it learns to generalize from the training data to generate accurate SQL queries for unseen prompts. The trained model and tokenizer are then saved, enabling seamless deployment for inference tasks.

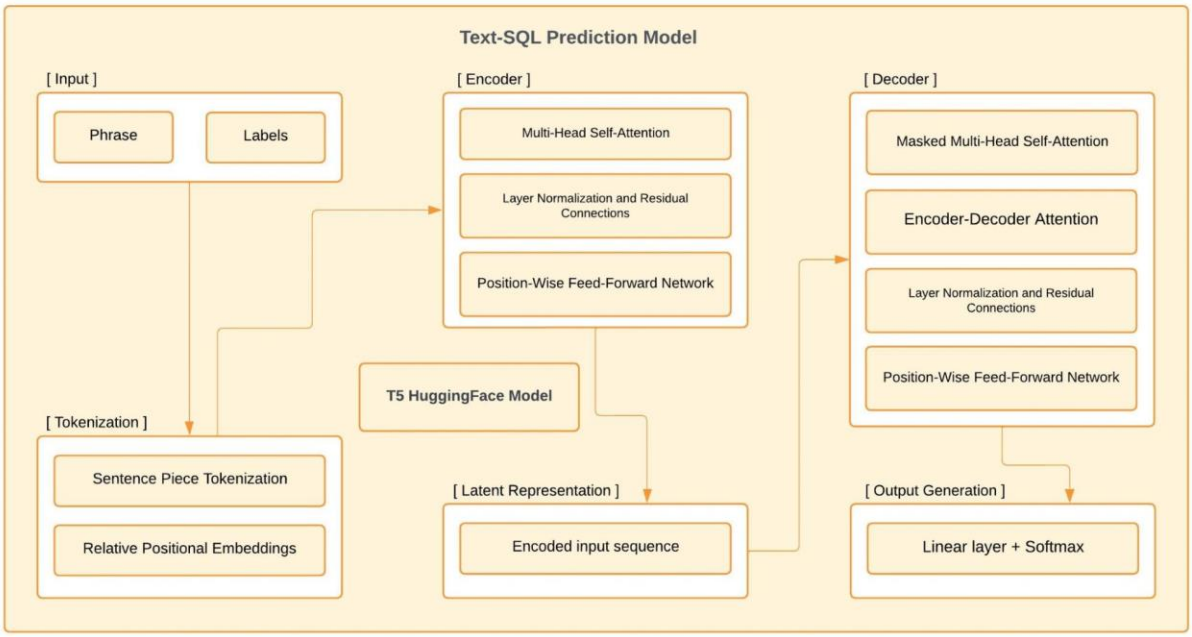

Fig. 6 Custom T5ForConditionalGeneration SQL Prediction Model

7. Results

The BERT-based multi-label classification model demonstrated strong performance in predicting SQL syntax labels. Over 20 epochs, the model achieved a training loss of 0.0844 and a test loss of 0.0706, highlighting its ability to generalize effectively to unseen data. It achieved a precision of 93.80%, balanced with a recall of 57.08%, resulting in an F1-score of 71.72%. Additional metrics, such as a subset accuracy of 75.12%, a Jaccard index of 84.12%, and a Hamming loss of 0.0227, further validated the model's robustness in handling multi-label classification tasks. These results emphasize BERT's ability to capture structural and semantic relationships between text prompts and SQL syntax components, making it a critical component in the overall pipeline.

```
------------------------------------------------
Epoch 15/20 | Train Loss: 0.0979, Time: 179.31s | Test Loss: 0.0809, Time: 48.86s | Precision: 0.9202,
Recall: 0.5136, F1-Score: 0.6735 | Subset Accuracy: 0.7455, Jaccard: 0.7870, Hamming Loss: 0.0248
------------------------------------------------
Epoch 16/20 | Train Loss: 0.0944, Time: 179.30s | Test Loss: 0.0782, Time: 48.87s | Precision: 0.9278,
Recall: 0.5385, F1-Score: 0.6948 | Subset Accuracy: 0.7500, Jaccard: 0.8140, Hamming Loss: 0.0242
------------------------------------------------
Epoch 17/20 | Train Loss: 0.0912, Time: 179.33s | Test Loss: 0.0761, Time: 48.89s | Precision: 0.9305,
Recall: 0.5550, F1-Score: 0.7043 | Subset Accuracy: 0.7502, Jaccard: 0.8252, Hamming Loss: 0.0240
------------------------------------------------
Epoch 18/20 | Train Loss: 0.0890, Time: 179.32s | Test Loss: 0.0740, Time: 48.85s | Precision: 0.9340,
Recall: 0.5610, F1-Score: 0.7098 | Subset Accuracy: 0.7509, Jaccard: 0.8310, Hamming Loss: 0.0235
------------------------------------------------
Epoch 19/20 | Train Loss: 0.0862, Time: 179.33s | Test Loss: 0.0723, Time: 48.87s | Precision: 0.9362,
Recall: 0.5655, F1-Score: 0.7135 | Subset Accuracy: 0.7510, Jaccard: 0.8364, Hamming Loss: 0.0232
------------------------------------------------
Epoch 20/20 | Train Loss: 0.0844, Time: 179.33s | Test Loss: 0.0706, Time: 48.87s | Precision: 0.9380,
Recall: 0.5708, F1-Score: 0.7172 | Subset Accuracy: 0.7512, Jaccard: 0.8412, Hamming Loss: 0.0227
------------------------------------------------
```

Fig. 7 Training Metrics of BERT Classification Model

The T5 Seq2Seq model, designed for natural language to SQL query generation, combines natural language prompts and SQL syntax labels to produce contextually accurate SQL queries. A similarity mapping process plays a vital role in redefining the user prompt to align with the database schema, ensuring clarity and relevance before being fed into the T5 model. This step uses techniques like tokenization, stemming, and fuzzy matching to resolve ambiguities, enabling the model to focus on schema-specific components. The similarity mapping ensures that user prompts are refined to match schema terms even if the input uses synonyms or alternate phrasing.

The T5 model was evaluated using the BLEU score, achieving a score of 56.8654, which highlights its effectiveness in generating SQL queries syntactically and semantically. Over 10 epochs, the model exhibited steady improvement, with training loss reducing from 0.7079 to 0.0145 and validation loss decreasing from 0.1236 to 0.0379. The combination of the T5 architecture and similarity mapping ensures the system can handle diverse user inputs and align them with database schema structures, resulting in precise SQL predictions.

```
BLEU Score on Validation Set: 56.8654
```

Fig. 7 BLEU Score Evaluation of T5 Model

```
Epoch 1: Train Loss: 0.7079, Val Loss: 0.1236
Epoch 2: Train Loss: 0.0772, Val Loss: 0.0612
Epoch 3: Train Loss: 0.0483, Val Loss: 0.0484
Epoch 4: Train Loss: 0.0369, Val Loss: 0.0432
Epoch 5: Train Loss: 0.0298, Val Loss: 0.0405
Epoch 6: Train Loss: 0.0250, Val Loss: 0.0387
Epoch 7: Train Loss: 0.0211, Val Loss: 0.0378
Epoch 8: Train Loss: 0.0182, Val Loss: 0.0383
Epoch 9: Train Loss: 0.0167, Val Loss: 0.0378
Epoch 10: Train Loss: 0.0145, Val Loss: 0.0379
```

Fig. 8 Training Metrics of T5 Generative model

Together, the BERT and T5 models, supported by the similarity mapping process, form a robust and complementary system for SQL generation. The BERT model extracts critical SQL syntax labels, while the T5 model, leveraging similarity mapping, generates accurate and executable SQL queries. This cohesive pipeline ensures high accuracy and adaptability, effectively translating natural language prompts into database queries.

8. Summary and Conclusion

The models developed in this project form an effective pipeline for translating natural language prompts into SQL queries. The BERT-based multi-label classification model accurately predicts SQL syntax labels, enabling structural understanding of queries, while the T5-based Seq2Seq model, supported by similarity mapping, generates contextually accurate SQL statements. Similarity mapping resolves ambiguities in user prompts by aligning them with database schema terms, ensuring that even non-standard phrasing leads to precise SQL generation. The models demonstrated robust performance, with the BERT model achieving high precision and recall for label prediction and the T5 model achieving a BLEU score of 56.8654, validating its capability in query generation.

Challenges Faced

Several challenges were encountered during the project. The label extraction process required careful design to ensure the model could effectively focus on SQL syntax components. Similarity mapping had to handle user inputs with synonyms or incomplete schema terms, which added complexity. Data preprocessing proved challenging when combining techniques like stemming, tokenization, and stop-word removal without losing critical semantic information.

Additionally, maintaining balance between training efficiency and model generalization required careful tuning of hyperparameters, particularly dropout rates and learning rates.

Future Enhancements

Future enhancements aim to make the system more robust and adaptable. Increasing the diversity of the training dataset, including more complex SQL syntaxes like JOIN, CASE, and UNION, will improve model recall and generalizability. Incorporating real-time adaptability for schema changes and enabling dynamic handling of varying database structures can further enhance scalability. Developing an intuitive user interface, potentially integrating voice-to-SQL capabilities, will make the system more accessible to non-technical users. Finally, optimizing model performance with advanced techniques, such as knowledge distillation or transfer learning, can reduce computational overhead while maintaining accuracy.

9. References

[1] - Pre-trained BERT base-uncased Model - Hugging Face
[2] - Pre-trained T5-small Model for Seq2Seq Tasks - Hugging Face
[3] - BLEU Score Metric - Medium
[4] - NLTK for Text Preprocessing - NLTK Official Site
[5] - WordNet for Synonym Generation - NLTK WordNet Interface
[6] - Fuzzy Matching Techniques - Python Library difflib
[7] - AdamW Optimizer - PyTorch Documentation
[8] - StepLR Learning Rate Scheduler - PyTorch Documentation
[9] - Multi-label Classification Metrics - Scikit-learn
[10] - Jaccard Index Metric - Scikit-learn Documentation
[11] - Hamming Loss Metric - Scikit-learn Documentation
[12] - Attention Mechanisms in Deep Learning - Medium
[13] - Sequence-to-Sequence Learning Overview - Arxiv