

**Final Report**

**Text-to-SQL Language Model**

Abdelghani Masad U21104769

Laith Mazen Kastali U21106049

Jamil Hatem El Bizri U21105497

Ahmad Allan U21107577

Omar Chikh U21106449

**College of Computing and Informatics**

**University of Sharjah**

**Topics in CS 2**

**April 2025**

Contents

[1.0 Introduction 3](#_Toc196169129)

[1.1 Problem Statement 3](#_Toc196169130)

[1.2 Project Objective 3](#_Toc196169131)

[2.0 Dataset 4](#_Toc196169132)

[3.0 Architecture 7](#_Toc196169133)

[4.0 Training and Parameters 8](#_Toc196169134)

[5.0 Conclusion 8](#_Toc196169135)

[5.1 Results 8](#_Toc196169136)

[5.2 Next Steps 9](#_Toc196169137)

[5.3 Conclusion 9](#_Toc196169138)

# Introduction

In today's data-driven world, the ability to extract meaningful insights from databases is crucial for decision-making across industries. However, interacting with databases requires knowledge of Structured Query Language (SQL), a skill that many non-technical users do not possess. This creates a significant barrier, especially for professionals in domains such as healthcare, business, and education, where quick access to data is essential but technical expertise may be limited.

## 1.1 Problem Statement

Despite the ubiquity of data, writing SQL queries remains a specialized task that often requires the assistance of data engineers or analysts. This dependency slows down workflows and limits data accessibility for many users. Bridging the gap between natural language and SQL is a powerful way to democratize data access, allowing users to query databases using plain English instead of learning complex syntax.

## 1.2 Project Objective

Our project addresses this challenge by developing a Text-to-SQL translation system powered by deep learning. Specifically, we implement a Sequence-to-Sequence (Seq2Seq) neural architecture with an attention mechanism, designed to convert natural language queries into syntactically correct SQL statements. The system utilizes transformer-based tokenization, learned embeddings, and a bidirectional encoder-decoder framework with attention to understand the semantics of user input and map it accurately to SQL syntax.

Ultimately, this solution has the potential to serve as a foundational tool in building intelligent, user-friendly database interfaces that empower users of all skill levels to interact with data effortlessly.

# 2.0 Dataset

A critical component of our Text-to-SQL system is the dataset used to train and evaluate the model. For this project, we curated a high-quality, preprocessed dataset derived from several well-established benchmarks, including **Kaggle and Hugging Face**. These datasets are widely recognized in the natural language to SQL research community and provide diverse examples of how human language maps to structured query formats.

To prepare the dataset for training, we undertook a comprehensive cleaning and preprocessing phase to ensure data quality and model readiness:

* **Null and empty entries** were identified and removed to avoid corrupt or meaningless training samples.
* We **dropped the schema column**, as our initial experiments focus on text-to-query translation without requiring schema awareness.
* The resulting dataset included only 425000 **pairs of natural language questions and their corresponding SQL queries**, allowing the model to learn precise mappings between the two modalities.

After preprocessing, the dataset was split into **90% for training** and **10% for validation**, ensuring a robust evaluation pipeline while maximizing training data availability. This split enables us to monitor overfitting and track generalization performance throughout the training process.

For tokenization, we used the **t5-small tokenizer** from HuggingFace's Transformers library. This tokenizer was selected for its ability to effectively handle both natural language and structured SQL syntax through subword tokenization. It preserves the integrity of SQL keywords while maintaining the flexibility needed to process diverse user inputs, which is essential for achieving accurate query generation.

A graph of a number of keywords

AI-generated content may be incorrect.

The bar chart titled **"Frequency of SQL Keywords"** reveals a striking imbalance in the distribution of SQL keywords within the dataset. The keywords **FROM**, **SELECT**, and **WHERE** dominate the dataset with exceptionally high frequencies, each appearing well over 400,000 times. In contrast, more advanced SQL operations such as **JOIN**, **GROUP BY**, **ORDER BY**, **AVG**, **SUM**, and **INSERT** occur significantly less frequently, while keywords like **DELETE** and **UPDATE** are nearly negligible.

This skew in keyword frequency indicates that the dataset is heavily focused on **simple, single-table queries**, which rely primarily on selecting and filtering data. While this simplifies initial model training and helps achieve high accuracy on straightforward questions, it introduces a serious limitation: the model may **underperform on complex, multi-table queries** that require less common SQL constructs. Consequently, the model might struggle to learn the semantic structure and syntactic patterns necessary for generating advanced SQL queries, such as those involving aggregations, nested subqueries, joins, or data modifications. For robust real-world deployment, it's essential to diversify the training data by including a richer mix of SQL operations that mirror practical, multi-faceted database use cases.

This carefully prepared dataset serves as the foundation of our model's learning, equipping it with the contextual patterns necessary to translate everyday language into valid SQL commands.

# 3.0 Architecture

At the heart of our Text-to-SQL system lies a custom-built **Sequence-to-Sequence (Seq2Seq)** neural architecture augmented with an **attention mechanism**. This architecture is designed to learn the complex mapping between natural language and structured SQL commands, even when faced with varying lengths, syntax structures, and semantic nuances.

The model is composed of four main components, each playing a pivotal role in the translation process:

**🔹 Encoder**

The encoder is a **bidirectional Gated Recurrent Unit (GRU)** network that processes the input question token by token. It begins by embedding each token using learned embeddings, which capture rich semantic information about each word or subword. The bidirectional design allows the encoder to consider both past and future context, ensuring that the model fully understands the meaning and intent behind each word in the input sequence. The output of the encoder is then passed through a linear transformation to align it with the decoder's expected input dimensions.

**🔹 Attention Mechanism**

The attention module serves as a bridge between the encoder and decoder, significantly enhancing the model’s performance. Rather than encoding the entire input into a single fixed-size vector (which can lead to information loss), attention allows the decoder to **dynamically focus** on different parts of the input sequence at each decoding step. This is especially useful in SQL generation, where specific keywords in the input need to map precisely to table names, conditions, or operations. The module computes attention scores through a feedforward network and uses them to generate context vectors—weighted summaries of the encoder outputs—tailored to each decoding step.

**🔹 Decoder**

The decoder is a **unidirectional GRU** that generates the SQL query token-by-token. At each time step, it takes three inputs: the embedding of the previously generated token, the current hidden state, and the context vector from the attention mechanism. By combining these elements, the decoder constructs a rich representation of the current state and predicts the next token in the SQL sequence. This design enables it to generate complex SQL queries with proper structure and logical flow.

**🔹 Seq2Seq Wrapper**

This wrapper module seamlessly integrates the encoder, attention mechanism, and decoder into a single cohesive framework. It manages the entire data flow—from encoding the input, calculating attention, to decoding the output. During training, it supports **teacher forcing**, a technique where the model is fed the correct previous token to guide learning. During inference (evaluation or deployment), it uses **greedy decoding** to generate the final SQL query based on the model's predictions.

This architecture strikes a balance between simplicity and expressiveness, making it capable of handling a wide variety of natural language inputs and translating them into precise SQL statements. Its modular design also allows for easy upgrades, such as incorporating schema-awareness or switching to transformer-based components in future iterations.

# 4.0 Training and Parameters

The training process was carefully designed to ensure both performance and reproducibility. We used a teacher-forced sequence-to-sequence training loop, supported by attention mechanisms, to optimize the model's ability to translate natural language into SQL. The model was trained over 8 epochs using the Adam optimizer and a cross-entropy loss function, while gradient clipping helped mitigate exploding gradients during backpropagation. To make training robust and repeatable, all relevant random number generators were seeded, and progress was regularly logged with validation checkpoints.

Key training configurations included:

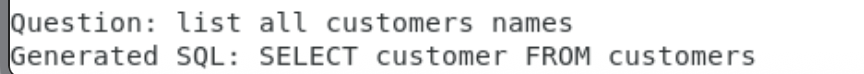
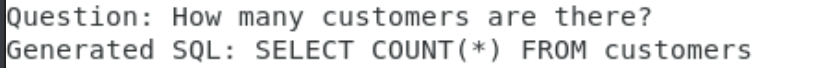
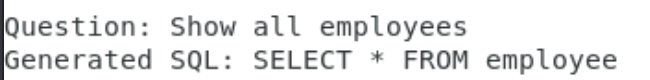
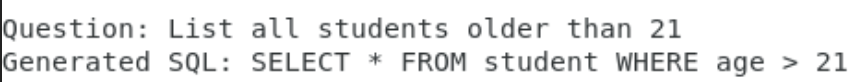
* **Embedding Dimension:** 256 – transforms tokens into dense vectors capturing semantic relationships.
* **Hidden Dimension:** 512 – used in the encoder and decoder GRU layers for deep sequence modeling.
* **Max Sequence Length:** 128 – ensures consistent input and output dimensions by truncating or padding sequences.
* **Batch Size:** 64 – balances memory efficiency and training stability.
* **Epochs:** 8 – chosen based on convergence observations.
* **Learning Rate:** 3e-4 – a moderate rate ensuring steady learning without overshooting.
* **Teacher Forcing Ratio:** 0.5 – helps stabilize training by mixing ground truth and model predictions during decoding.
* **Gradient Clipping:** 1.0 – prevents exploding gradients by capping the norm of gradients.
* **Optimizer:** Adam – known for adaptive learning rates and effective convergence.
* **Loss Function:** CrossEntropyLoss – used while ignoring padding tokens to avoid penalizing the model for padded outputs.

The model was trained using a custom loop that evaluated validation performance after each epoch, saved the best-performing model based on validation loss, and tested real-time SQL generation on a set of sample questions.

## 5.1 Results

After training our Seq2Seq model with attention for **8 epochs**, we observed significant improvements in both training and validation loss, indicating that the model was learning to generalize effectively. Most importantly, the model began generating **syntactically and semantically correct SQL queries** in response to diverse natural language inputs.

Here are some example outputs that reflect the model’s capability:

****

These examples demonstrate the model’s proficiency in understanding user intent, applying simple filters, and producing correctly structured SQL queries—even for inputs with varying complexity. The attention mechanism played a vital role in aligning tokens and capturing relationships between entities, conditions, and commands within the queries.

## 5.2 Next Steps

To enhance the system’s accuracy, versatility, and real-world usability, we have identified several important directions for future work:

1. **Schema-aware SQL Generation**  
   Integrate database schema information into the model to generate more accurate and context-aware queries. This will enable handling of multi-table operations, foreign key relations, and specific column selections.
2. **Fine-tuning with Complex Datasets (Spider, CoSQL)**  
   Extend training on advanced datasets such as Spider and CoSQL, which include complex, nested, and multi-domain queries. This step will improve the model's performance in handling real-world scenarios and increase generalizability.
3. **Web-based Deployment**  
   Develop an intuitive web application where users can input questions and receive instant SQL query responses. This interface can be integrated with a backend database for real-time querying and visualization.

## 5.3 Conclusion

In this project, we developed a text-to-SQL system using a Seq2Seq architecture with attention and the T5 tokenizer, enabling natural language questions to be translated into accurate SQL queries. Through this process, we gained valuable insights into neural machine translation, attention mechanisms, and learned embeddings.

This work is highly useful in making databases more accessible to non-technical users by bridging the gap between human language and structured query languages. With further refinement, such systems can support real-world applications in data analysis, customer support, business intelligence, and beyond—empowering users to interact with data more naturally and efficiently.