Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Natural Language Database Querying Based on Deep Learning

July 2022

Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Natural Language Database Querying Based on Deep Learning

By:

Marwan Mohamed Mostafa Mohi ElDeen [CS]
Mahmoud Adel Kamal Ismail [CS]
Mahmoud Alaa Morsy Abdeltawab [CS]
Mahmoud Mohamed Saeed Mohamed Sobhy [CS]

Under Supervision of :

Dr. Dina Khattab
SC
Faculty of Computer and Information Sciences,
Ain Shams University.

T.A Marwah Helaly
CS
Faculty of Computer and Information Sciences,
Ain Shams University.

Table Of Content

Acknowledgement

All praise and thanks to ALLAH for being with us along this journey and giving us the power to finish it and rewarding us for this hard work.

We would like to thank Dr. Dina Khattab, our project supervisor, for her patient instruction, passionate support, and constructive criticisms of this project effort.

We would also like to thank T.A Marwah Helaly for her wonderful guidance and her fervent assistance in keeping our development on the right track.

Abstract

Nowadays, data represents an essential demand in a lot of domains including education, business and healthcare. Recently, these data are stored in databases and controlled, manipulated and updated by DBMS.

Therefore, databases represent a great source of information. The number of databases as well as their size and complexity are increasing. To extract information from these databases, the user needs to write queries using database query languages, such as structured query language (SQL). This creates a barrier to use especially for non-experts, who must come to grips with the nature of the data, the way it has been represented in the database, and the specific query languages or user interfaces by which data is accessed. These difficulties worsen in research settings, where it is common to work with many different databases.

From here our idea is born, what are the easiest ways to extract your data from databases? Of course, one of them is to ask for it with a simple question represented in your natural language. Our mission is to transform this simple question into a database query through an AI model and run this query on the database then retrieve and display the requested information to the user.

For example, in health care. Medical staff can retrieve important info about their patients, compare different cases, and get all the important information instantly and easily. Scientists in pharmaceutical laboratories can manipulate and retrieve important data with just a few words represented in a simple natural language query. Thus, there are many scenarios that can be easily resolved without any difficulties.

List of Figures

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations From Transformers
CPU	Central Processing Unit
CNN	Convolutional Neural Networks
ConvS2S	Convolutional sequence to sequence
DB	DataBase
e.g.	example
GPU	Graphics processing unit
GPT3	Generative Pre-trained Transformer 3
IDE	Integrated Development Environment
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
NER	Named Entity Recognition
NMT	Neural Machine Translation
NL	Natural Language
NLKBIDB	Natural Language and keyword based interface to database
NLP	Natural Language Processing
NLQ	Natural Language Querying
NLTK	Natural Language Toolkit
NN	Neural Networks
RNN	Recurrent Neural Network
SQL	Structured Query Language

Introduction

1.1 Motivation

Most of the media networks and organizations require a database to store the information and to retrieve the information from the database where structured query language (SQL) is utilized. Our project resolves problems faced by almost all domains in this field as most domains need to store their data and access this data at any time.

The current solution to access your data from databases is to hire a software engineer to make a program for you with special filters that allow you to manipulate your data but this approach is not efficient enough as you probably want information with complex constraints which are not covered with only filters. Also our approach is more time efficient as you should only ask for the info you want with your simple natural language without the headache of filters.

1.2 Problem Definition

Nowadays, data represents an essential demand in a lot of domains. Recently, these data are stored in databases. The number of databases as well as their size and complexity increases, which makes the process of manipulating the data from them a more challenging task. So, we aim to translate natural language questions asked by non-experts to database queries, and then retrieve the answer from the database records.

1.3 Objective

Our aim is to develop a web application that:

- Enables non-professional users to easily retrieve the data they need from the databases.
- Accelerates the process of writing complex queries for professional users.
- Supports multiple domains e.g Education, HealthCare, Business and more.

1.4 Time Plan

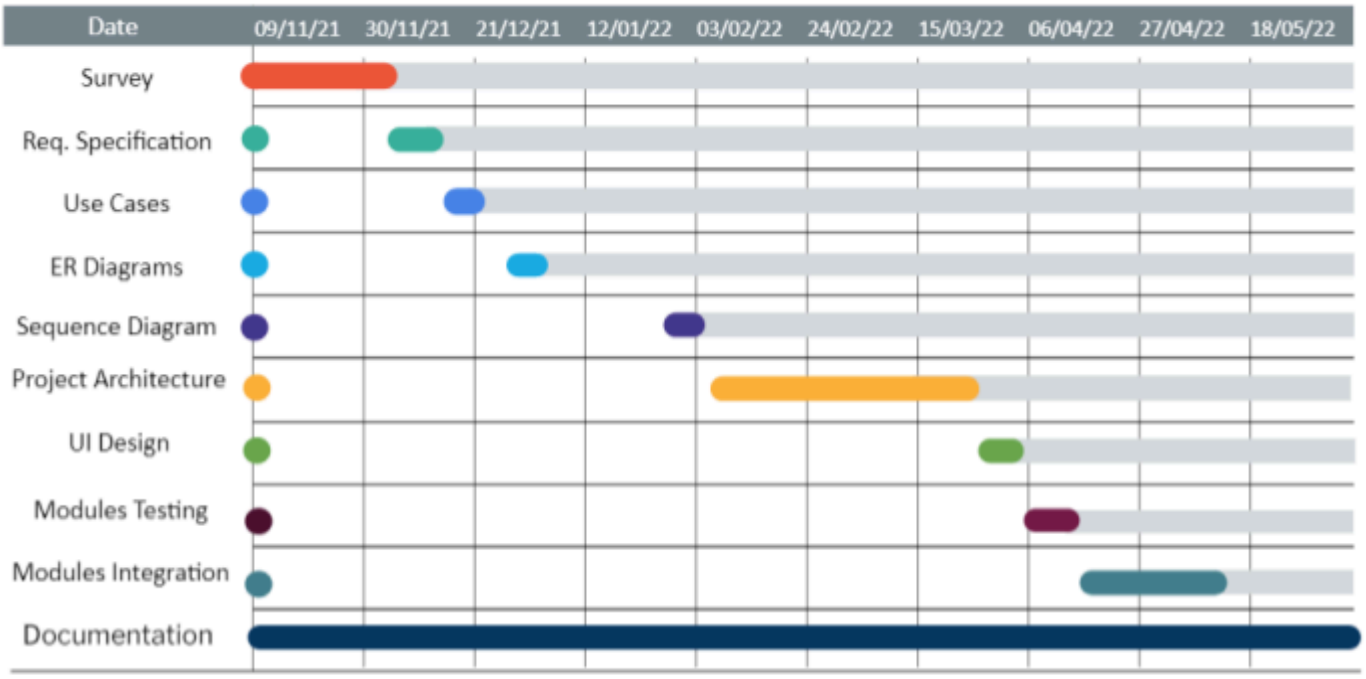

Figure 1.1. Time Plan

1.5 Document Organization

Chapter 2: Background

This chapter contains a detailed description of the field of the project, all the scientific background related to the project, a survey of the work done in the field and description of existing similar systems.

Chapter 3: Analysis and Design

This chapter describes the system architecture and how it communicates with internal and external modules.

Chapter 4: Implementation

This chapter includes a detailed description of all the functions in the system, A detailed description of all the techniques and algorithms implemented, and Description of any new technologies used in implementation.

Chapter 5: User Manual

This chapter describes in detail how to operate the project along with screen shots of the project representing all steps.

Chapter 6: Conclusions and Future Work

This chapter contains a complete summary of the project and how we would be able to improve the performance and what are new features that can be added in the future.

Background

2.1 Neural Networks

Neural networks are a set of algorithms, modeled loosely after the human brain, that is designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling, or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text, or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on.

The researchers considered the neural network as a black box strategy as shown in Figure 2.1, which is trainable.

Figure 2.1. Neural Network Black-Box

The key aspect of black-box approaches is developing relationships between input and output. The researchers tried to 'train' the neural black-box to 'learn' the correct response output for each of the training samples[X].

2.1.1 Neural Network Elements

Deep learning is the name we use for "stacked neural networks"; that is, networks composed of several layers.
The layers are made of nodes. A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs with regard to the task the algorithm is trying to

learn; e.g. which input is most helpful is classifying data without error? These input-weight products are summed. The sum is passed through a node's so-called activation function, to determine whether and to what extent that signal should progress further. Say, an act of classification. If the signal passes through, the neuron has been "activated."

As shown in Figure 2.2 this is what one node might look like.

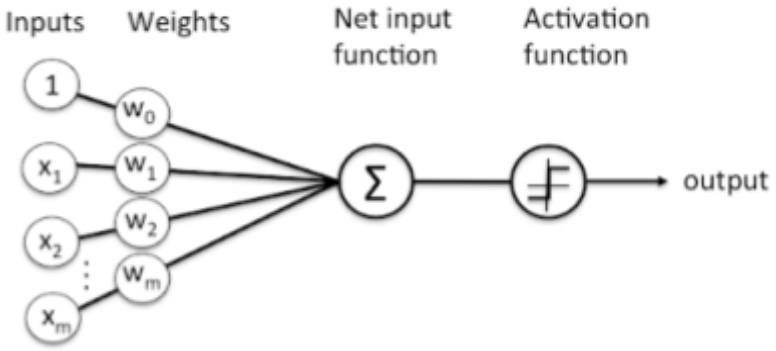

Figure 2.2. Diagram for one node

A node layer is a row of those neuron-like switches that turn on or off as the input is fed through the net. Each layer's output is simultaneously the subsequent layer's input, starting from an initial input layer receiving your data as shown in Figure 2.3.

Figure 2.3. Neural Network

Pairing the model's adjustable weights with input features is how we assign significance to those features with regard to how the neural network classifies and clusters input.

2.1.2 Problems Commonly Solved With Neural Networks

There are many different problems that can be solved with a neural network. However, neural networks are commonly used to address particular types of problems. The following types of problem are frequently solved with neural networks:

- Regression.
- Classification.
- Pattern recognition.
- Prediction.
- Optimization.
- Clustering.

2.1.3 Main characteristics of Neural Network

1. Activation function:
   A function that produces an output based on the input values received by node like:
   1. Sigmoid.
   2. ReLU.
   3. Hyperbolic Tangent.
2. Architecture or structure:
   The connectivity of neurons (nodes) determines the neural network structure (architecture).
3. The learning algorithm, or training method:
   Method for determining weights of the connections. The manner in which the neurons of neural networks are structured is intimately linked with the learning algorithm used to train the network.

2.2 Natural Language Processing "NLP"

Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken and written -- referred to as natural language. It is a component of artificial intelligence (AI).

NLP has existed for more than 50 years and has roots in the field of linguistics. It has a variety of real-world applications in a number of fields, including medical research, search engines and business intelligence.

2.2.1 How does NLP work?

NLP enables computers to understand natural language as humans do. Whether the language is spoken or written, natural language processing uses artificial intelligence to take real-world input, process it, and make sense of it in a way a computer can understand. Just as humans have different sensors -- such as ears to hear and eyes to see -- computers have programs to read and microphones to collect audio. And just as humans have a brain to process that input, computers have a program to process their respective inputs. At some point in processing, the input is converted to code that the computer can understand.

There are two main phases to natural language processing: data preprocessing and algorithm development.

Data preprocessing involves preparing and "cleaning" text data for machines to be able to analyze it. preprocessing puts data in workable form and highlights features in the text that an algorithm can work with. There are several ways this can be done, including:

- **Tokenization.** This is when text is broken down into smaller units to work with.
- **Stop word removal.** This is when common words are removed from text so unique words that offer the most information about the text remain.
- **Lemmatization and stemming.** This is when words are reduced to their root forms to process.
- **Part-of-speech tagging.** This is when words are marked based on the part-of-speech they are -- such as nouns, verbs and adjectives.

Once the data has been preprocessed, an algorithm is developed to process it. There are many different natural language processing algorithms, but two main types are commonly used:

- **Rules-based system.** This system uses carefully designed linguistic rules. This approach was used early on in the development of natural language processing, and is still used.

- **Machine learning-based system.** Machine learning algorithms use statistical methods. They learn to perform tasks based on training data they are fed, and adjust their methods as more data is processed. Using a combination of machine learning, deep learning and neural networks, natural language processing algorithms hone their own rules through repeated processing and learning.

2.2.2 Techniques and methods of NLP

Syntax and semantic analysis are two main techniques used with natural language processing.

Syntax is the arrangement of words in a sentence to make grammatical sense. NLP uses syntax to assess meaning from a language based on grammatical rules. Syntax techniques include:

- **Parsing.** This is the grammatical analysis of a sentence. *Example*: A natural language processing algorithm is fed the sentence, "The dog barked." Parsing involves breaking this sentence into parts of speech -- i.e., dog = noun, barked = verb. This is useful for more complex downstream processing tasks.
- **Word segmentation.** This is the act of taking a string of text and deriving word forms from it. *Example:* A person scans a handwritten document into a computer. The algorithm would be able to analyze the page and recognize that the words are divided by white spaces.
- **Sentence breaking.** This places sentence boundaries in large texts. *Example:* A natural language processing algorithm is fed the text, "The dog barked. I woke up." The algorithm can recognize the period that splits up the sentences using sentence breaking.
- **Morphological segmentation.** This divides words into smaller parts called morphemes. *Example:* The word untestably would be broken into [[un[[test]able]]ly], where the algorithm recognizes "un," "test," "able" and "ly" as morphemes. This is especially useful in machine translation and speech recognition.
- **Stemming.** This divides words with inflection in them to root forms. *Example:* In the sentence, "The dog barked," the algorithm

would be able to recognize the root of the word "barked" is "bark." This would be useful if a user was analyzing a text for all instances of the word bark, as well as all of its conjugations. The algorithm can see that they are essentially the same word even though the letters are different.

Semantics involves the use of and meaning behind words. Natural language processing applies algorithms to understand the meaning and structure of sentences. Semantics techniques include:

- **Word sense disambiguation.** This derives the meaning of a word based on context. *Example:* Consider the sentence, "The pig is in the pen." The word pen has different meanings. An algorithm using this method can understand that the use of the word *pen* here refers to a fenced-in area, not a writing implement.
- **Named entity recognition.** This determines words that can be categorized into groups. *Example:* An algorithm using this method could analyze a news article and identify all mentions of a certain company or product. Using the semantics of the text, it would be able to differentiate between entities that are visually the same. For instance, in the sentence, "Daniel McDonald's son went to McDonald's and ordered a Happy Meal," the algorithm could recognize the two instances of "McDonald's" as two separate entities -- one a restaurant and one a person.
- **Natural language generation.** This uses a database to determine semantics behind words and generate new text. *Example:* An algorithm could automatically write a summary of findings from a business intelligence platform, mapping certain words and phrases to features of the data in the BI platform. Another example would be automatically generating news articles or tweets based on a certain body of text used for training.

Current approaches to natural language processing are based on deep learning, a type of AI that examines and uses patterns in data to improve a program's understanding. Deep learning models require massive amounts of labeled data for the natural language processing algorithm to train on

and identify relevant correlations, and assembling this kind of big data set is one of the main hurdles to natural language processing.

Earlier approaches to natural language processing involved a more rules-based approach, where simpler machine learning algorithms were told what words and phrases to look for in text and given specific responses when those phrases appeared. But deep learning is a more flexible, intuitive approach in which algorithms learn to identify speakers' intent from many examples -- almost like how a child would learn human language.

Three tools used commonly for natural language processing include Natural Language Toolkit (NLTK), Gensim and Intel natural language processing Architect. NLTK is an open source Python module with data sets and tutorials. Gensim is a Python library for topic modeling and document indexing. Intel NLP Architect is another Python library for deep learning topologies and techniques.

2.2.3 Applications of NLP

Some of the main functions that natural language processing algorithms perform are:

- **Text classification.** This involves assigning tags to texts to put them in categories. This can be useful for sentiment analysis, which helps the natural language processing algorithm determine the sentiment, or emotion behind a text. For example, when brand A is mentioned in X number of texts, the algorithm can determine how many of those mentions were positive and how many were negative. It can also be useful for intent detection, which helps predict what the speaker or writer may do based on the text they are producing.
- **Text extraction.** This involves automatically summarizing text and finding important pieces of data. One example of this is keyword extraction, which pulls the most important words from the text, which can be useful for search engine optimization. Doing this with natural language processing requires some programming -- it is not completely automated. However, there are plenty of simple keyword extraction tools that automate most of the process -- the

user just has to set parameters within the program. For example, a tool might pull out the most frequently used words in the text. Another example is named entity recognition, which extracts the names of people, places and other entities from text.

- **Machine translation.** This is the process by which a computer translates text from one language, such as English, to another language, such as French, without human intervention.
- **Natural language generation.** This involves using natural language processing algorithms to analyze unstructured data and automatically produce content based on that data. One example of this is in language models such as GPT3, which are able to analyze an unstructured text and then generate believable articles based on the text.

The functions listed above are used in a variety of real-world applications, including:

- customer feedback analysis -- where AI analyzes social media reviews;
- customer service automation -- where voice assistants on the other end of a customer service phone line are able to use speech recognition to understand what the customer is saying, so that it can direct the call correctly;
- automatic translation -- using tools such as Google Translate, Bing Translator and Translate Me;
- academic research and analysis -- where AI is able to analyze huge amounts of academic material and research papers not just based on the metadata of the text, but the text itself;
- analysis and categorization of medical records -- where AI uses insights to predict, and ideally prevent, disease;
- word processors used for plagiarism and proofreading -- using tools such as Grammarly and Microsoft Word;
- stock forecasting and insights into financial trading -- using AI to analyze market history and 10-K documents, which contain comprehensive summaries about a company's financial performance;
- talent recruitment in human resources; and

- automation of routine litigation tasks -- one example is the artificially intelligent attorney.

Research being done on natural language processing revolves around search, especially Enterprise search. This involves having users query data sets in the form of a question that they might pose to another person. The machine interprets the important elements of the human language sentence, which correspond to specific features in a data set, and returns an answer.

NLP can be used to interpret free, unstructured text and make it analyzable. There is a tremendous amount of information stored in free text files, such as patients' medical records. Before deep learning-based NLP models, this information was inaccessible to computer-assisted analysis and could not be analyzed in any systematic way. With NLP analysts can sift through massive amounts of free text to find relevant information.

Sentiment analysis is another primary use case for NLP. Using sentiment analysis, data scientists can assess comments on social media to see how their business's brand is performing, or review notes from customer service teams to identify areas where people want the business to perform better.

2.3 Transformer

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU, ByteNet and ConvS2S , all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention, Self-attention, sometimes called intra-attention is an attention mechanism relating

different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations. End-to-end memory networks are based on a recurrent attention mechanism instead of sequence aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks. To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution.

2.3.1 Model Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure. Here, the encoder maps an input sequence of symbol representations (x1, ..., xn) to a sequence of continuous representations z = (z1, ..., zn). Given z, the decoder then generates an output sequence (y1, ..., ym) of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

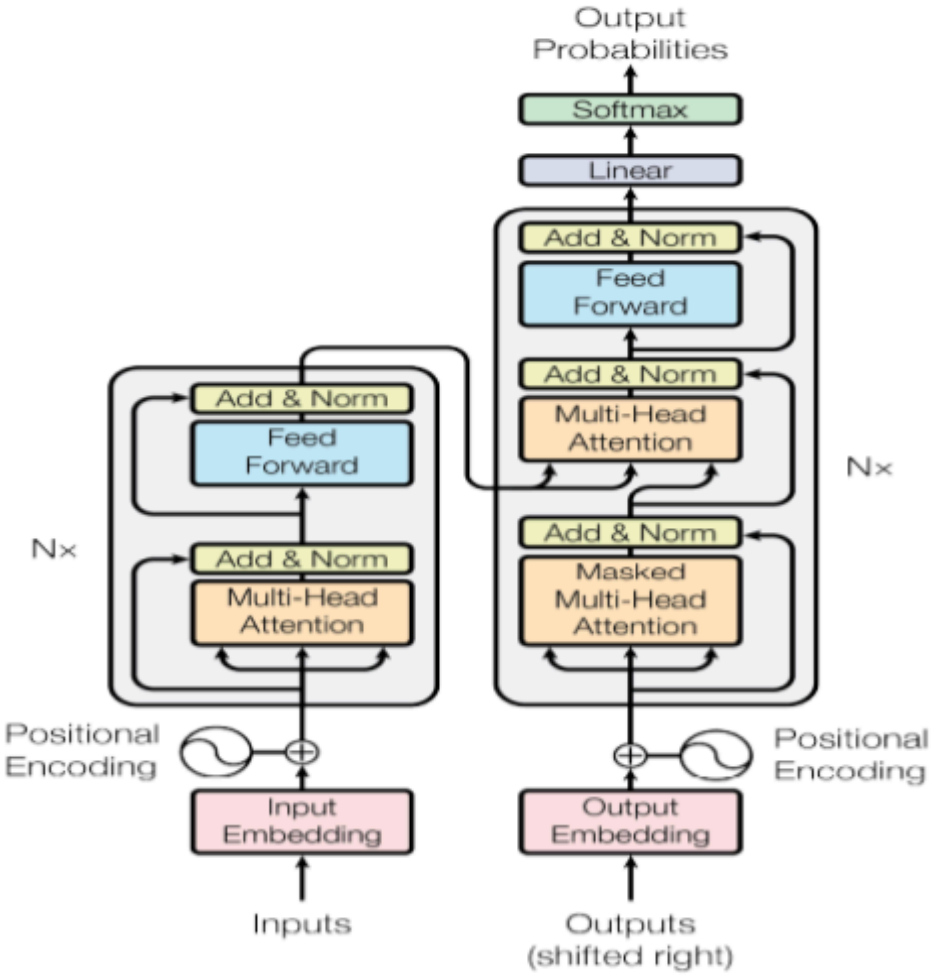

Figure 2.4. The Transformer - model architecture

2.3.2 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of N = 6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension dmodel = 512.

Decoder: The decoder is also composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder

inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

2.3.3 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

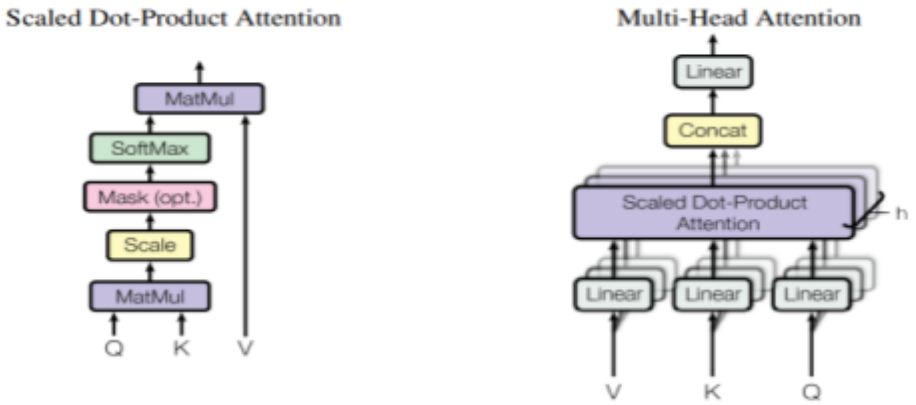

Figure 2.5. (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel

2.3.3.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension dk, and values of dimension dv. We compute the dot products of the query with all keys, divide each by $\sqrt{dk}$, and apply a softmax function to obtain the weights on the values. In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and

values are also packed together into matrices K and V . We compute the matrix of outputs as:

$$Attention(Q,\ K,\ V\ )\ =\ softmax(\frac{QKT}{\sqrt{dk}}\ )V\ (1)$$

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{dk}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. While for small values of dk the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of dk [3]. We suspect that for large values of dk, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients 4 . To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{dk}}$.

2.3.3.2 Multi-Head Attention

Instead of performing a single attention function with dmodel-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to dk, dk and dv dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding dv-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

2.3.3.3 Applications of Attention in our Model

The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

2.4 Natural language Interface

Natural language interfaces aim at integrating natural language processing and human-computer interactions. It seeks to provide means for humans to interact with computers through the use of natural language. We explore a specific aspect of the research which applies to relational databases which can be essentially

summarized as translation of natural language questions asked by the user to corresponding SQL queries. Neural machine translation (NMT) is an approach to machine translation that uses an artificial neural network to predict the likelihood of a sequence of words, typically modeling entire sentences in a single integrated model. Deep learning applications appeared first in speech recognition in the 1990s. The research on using neural networks for machine translation started gaining momentum in 2014, when Sutskever et al. (2014) published a paper on sequence to sequence learning with neural networks. Other work using encoder-decoder networks include Cho et al. (2014), Bahdanau et al. (2014), Wu et al. (2016). Variants of seq2seq were used by Vinyals et al. (2015) and Dong and Lapata (2016) to predict next characters in the sequence.

2.5 Survey

Natural language querying (NLQ) is a known problem in information retrieval. It allows questions to be formed without knowledge of database-specific logical languages such as SQL or Cypher. In principle this can ease data access for non-expert users. To this end, several approaches to build NLQ systems have been proposed. Recent surveys, segmented them into five approaches: keyword-based, pattern-based, syntax-based, grammar-based, and, more recently, connectionist-based.

In keyword-based systems, the approach has two stages, a first stage where keywords present in an input query are extracted, and a second stage where keywords are matched against metadata available in the underlying database. For instance, Blunschi et al. described Search over DAta Warehouse (SODA), which generates SQL queries from natural language (NL) queries over a business-related database. The system processes an input NL query through a series of steps. First, the keywords in the query are matched against all the possible entries in the database metadata. Second, by means of a heuristic, each result is scored and the process continues with the top N results. A third step identifies the tables used by each result and their relationships before being passed to the fourth step, which is responsible for finding, from the original query, the needed filters over the tables and columns. Last, the gathered information from the previous steps is combined to generate a SQL query that takes into account possible join patterns by looking at foreign keys and inheritance patterns in the schema. A drawback of this approach is it

requires hand-crafting the patterns that are used to translate from a keyword-based input to a SQL query for the specific modelling of the target database, which becomes a bottleneck when trying to use it across a variety of databases.

In pattern-based systems, the capabilities of keyword-based approaches are extended by including NL patterns for processing queries, alleviating issues such as aggregation operations. For instance, Shah et al described NLKBIDB, a NL to SQL query interface that uses NL patterns to fix syntactically incorrect queries, and a keyword-based approach to obtain the corresponding facts from the schema before carrying out the conversion. The system uses lexical analysis to tokenize the NL query and syntax analysis to parse the lexicons. If the input query is syntactically valid, then the lexicons are analyzed by a semantic analyzer using a domain ontology before generating a SQL query. Invalid queries are converted into SQL by applying a set of rules. An issue that arises from this approach is the need for a knowledge expert to refine the rules used to convert syntactically invalid queries into SQL sentences, as well as the need to use hand-crafted natural language rules for generating the SQL queries for syntactically valid queries. This makes it difficult to adapt for new use cases. In ATHENA++ is presented to cope with nested SQL queries. In this approach linguistic patterns from NL queries are combined with domain reasoning using ontologies to enable nested query detection and generation. In addition, a new benchmark dataset (FIBEN) is introduced, containing 237 distinct complex SQL queries on a database with 152 tables. This approach tries to translate a NL query to OQL using the domain ontology, and then the OQL query is translated to SQL by using the mappings between the ontology and the database. It appears that one potential limitation of such an approach is the requirement to have both a domain ontology and a defined mapping of that ontology to the database.

Syntax-based and grammar-based systems[7] share similar methodologies. In both approaches, the NL query is parsed using linguistic rules to produce a syntax tree. Then, for syntax-based systems, the concepts in the tree are mapped to a query in the target query language (e.g. SQL). This has a variety of issues. First, a given query may have different parse trees, which after mapping may produce different queries.

Another issue is deciding which concepts from the query to map and which ones not. To alleviate this, grammar-based systems exploit domain knowledge captured by the grammar with the aim of reducing ambiguity when mapping queries. However, this method requires knowledge of the domain to build an effective parsing grammar, making it hard to adapt to new domains.

In contrast to approaches treating items in a language as symbols (symbolic approaches), relying on theoretical foundations in linguistics, connectionist approaches, also known as computational intelligence-based approaches, try to learn statistical patterns in the data and/or distributed representations of it, allowing for a richer linguistic variability. These methods can be divided into traditional machine learning and more recent deep learning techniques. The latter have become widely used in natural language processing (NLP) tasks, achieving state-of-the-art results. The main difference between these methods and traditional machine learning is that their objective is to learn a distributed representation (in the form of real-valued vectors) of the data, without the need for the feature engineering stage that earlier approaches required.

2.5 Description of existing similar systems

2.5.1 Natural Language User Interface for SAP

NLSQL converts Natural Language request to SQL queries and get's back with text response and SQL script via API web service. It helps SAP administrators to provide SAP users with Natural Language Interface for SAP. Natural Language User Interface for SQL API provides the possibility for End Users to request information from SAP database using the only Natural Language. for more info you can visit : **store.sap.com**

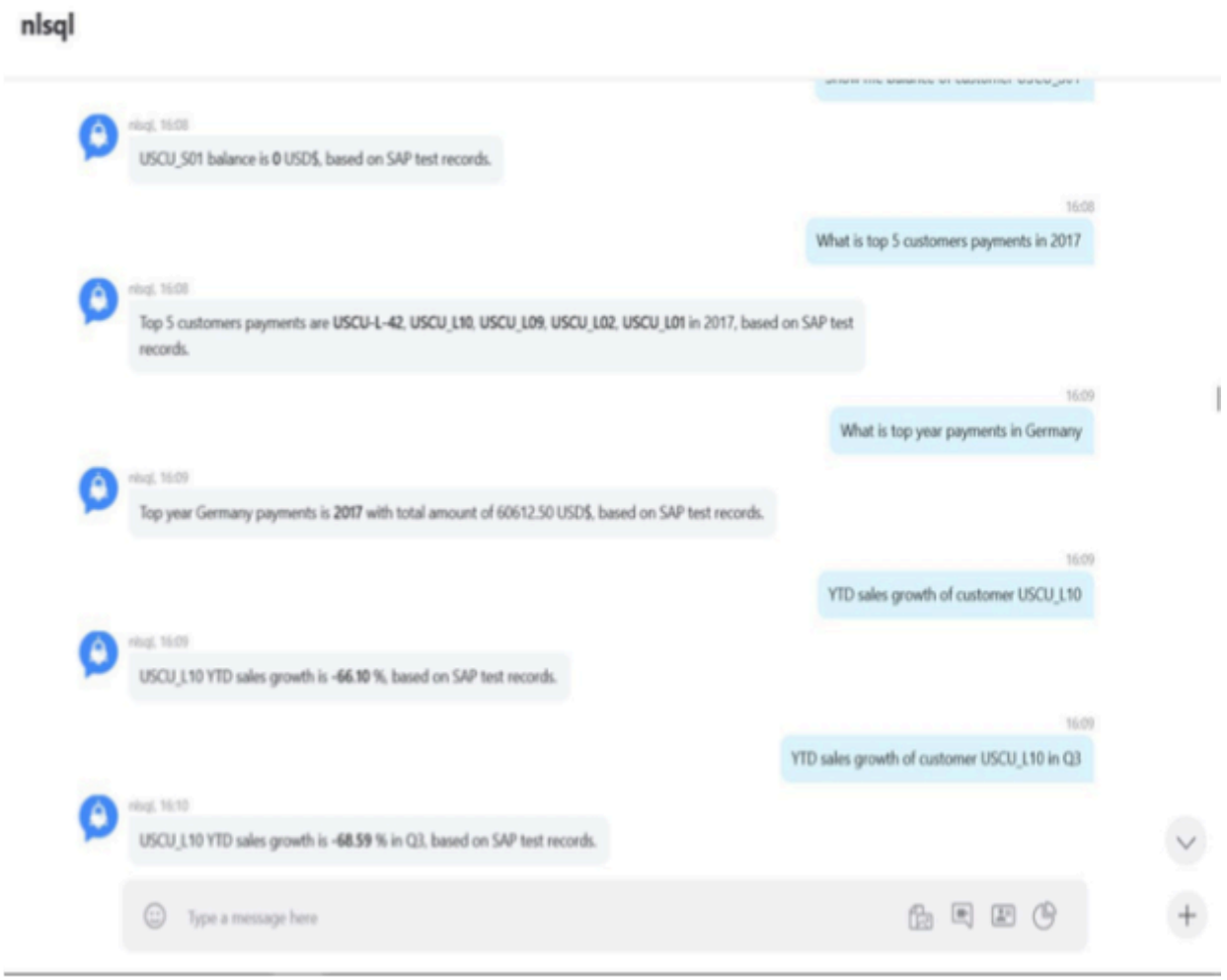

Figure 2.6. NLSQL

2.5.2 CHATA

Chata embeds an intelligent, conversational interface that allows users to access and analyze data simply by typing requests in everyday language. for more info you can visit : **chata.ai**

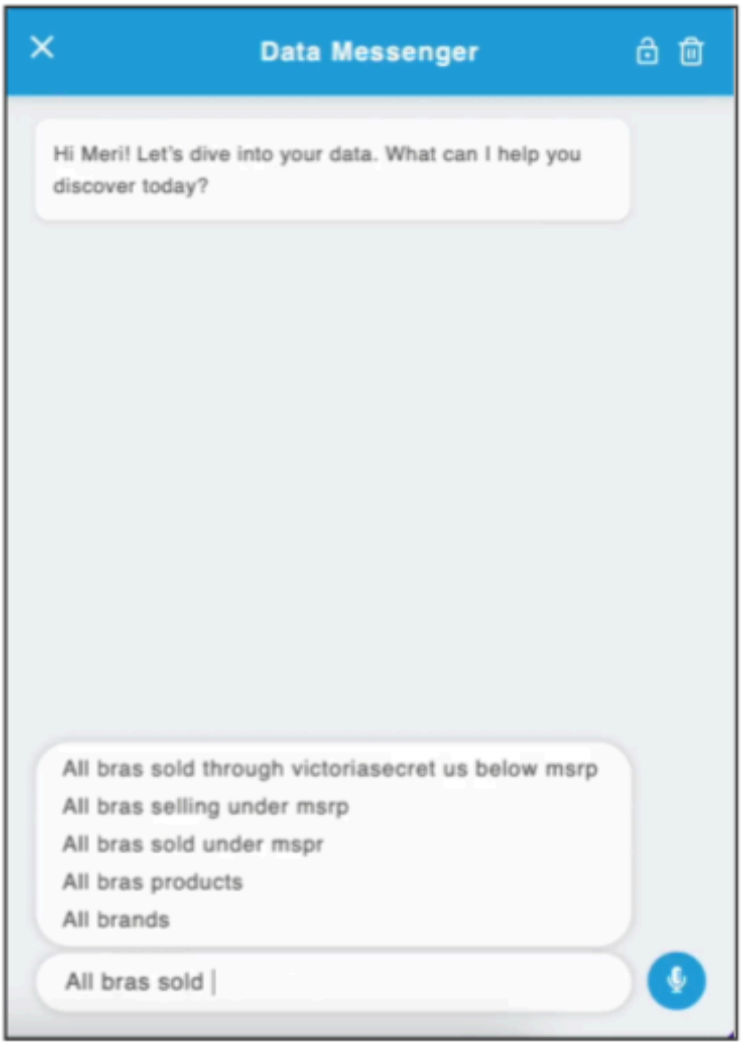

Figure 2.7. CHATA

1- Analysis and Design

3.1 System Overview

3.1.1 System Architecture

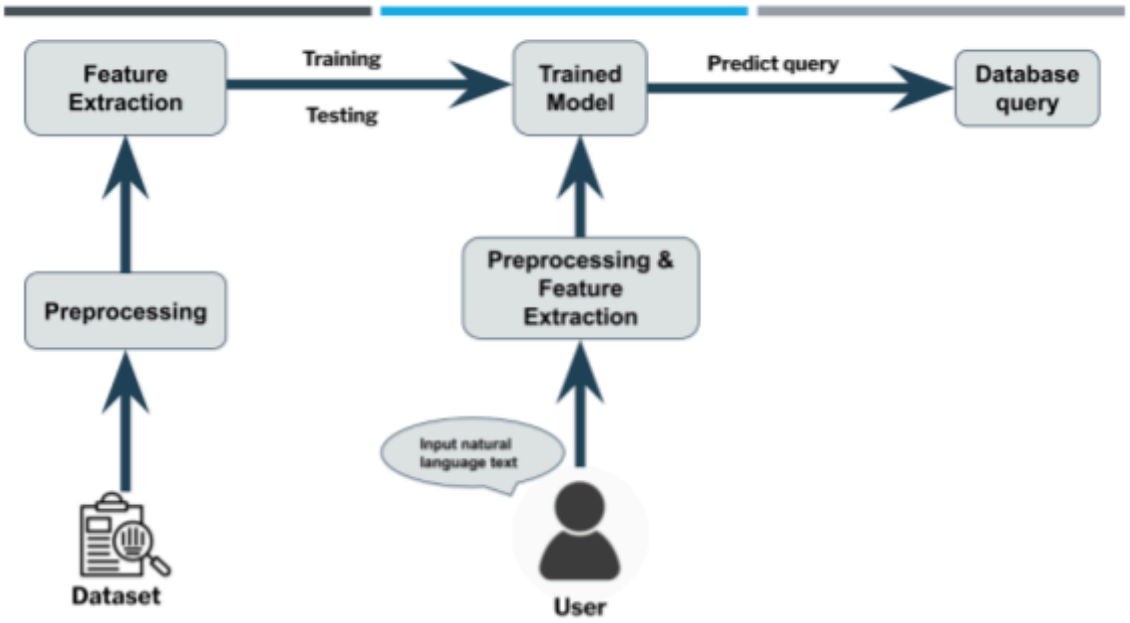

Figure 3.1. System Architecture

1. Selecting Dataset: We selected Spider dataset as it is a large-scale complex and cross-domain text-to-SQL dataset annotated by Yale University students.

2. Preprocessing Dataset: The process of cleaning and preparing the raw data to enable feature extraction.

3. Feature Extraction: Preparing the features of the data for the ML model

4. Splitting dataset: Dividing the data into two (or more) subsets. The first part is used to train the model and the other is used to evaluate the model.

5. Model building & training: Creating a model, training and evaluating it, and repeating the previous steps till we build a model that achieve good accuracy

6. Preprocessing user Input: Preparing the user input for the ML model so that the model is able to easily interpret the input data features.

7. Predicting query: Running the trained model on the input data and returning the output database query.

A. Functional Requirements
   - The system should take an English input text from the user.
   - The system should be able to generate SQL query from the input.
   - The system should be able to notify the user with the progress while generating the SQL query.
   - The user should be able to choose the desired database.

B. non-Functional Requirements
   - The system should be efficient in using computer resources.
   - Users with no training shall be able to use the system.
   - The system shall be self‑explanatory and intuitive.
   - The user interface should be efficient and easy to use.
   - The translated SQL query should be close enough to the meaning text input said by the user.

3.1.2 System Users

A. Intended Users:

1. Database Engineers:

- *Their main job is to write SQL queries to retrieve data from the database.*
- *The system will make their jobs easier by using an automated tool to generate the SQL query for them.*

2. Regular Users:

- *The system will help them to get the SQL query to execute it on the database.*

B. User Characteristics
- *Knowledge of computers (any person who can use computers).*
- *Understanding how to run the SQL query on the database management tool.*

3.2 System Analysis & Design

3.2.1 Use Case Diagram

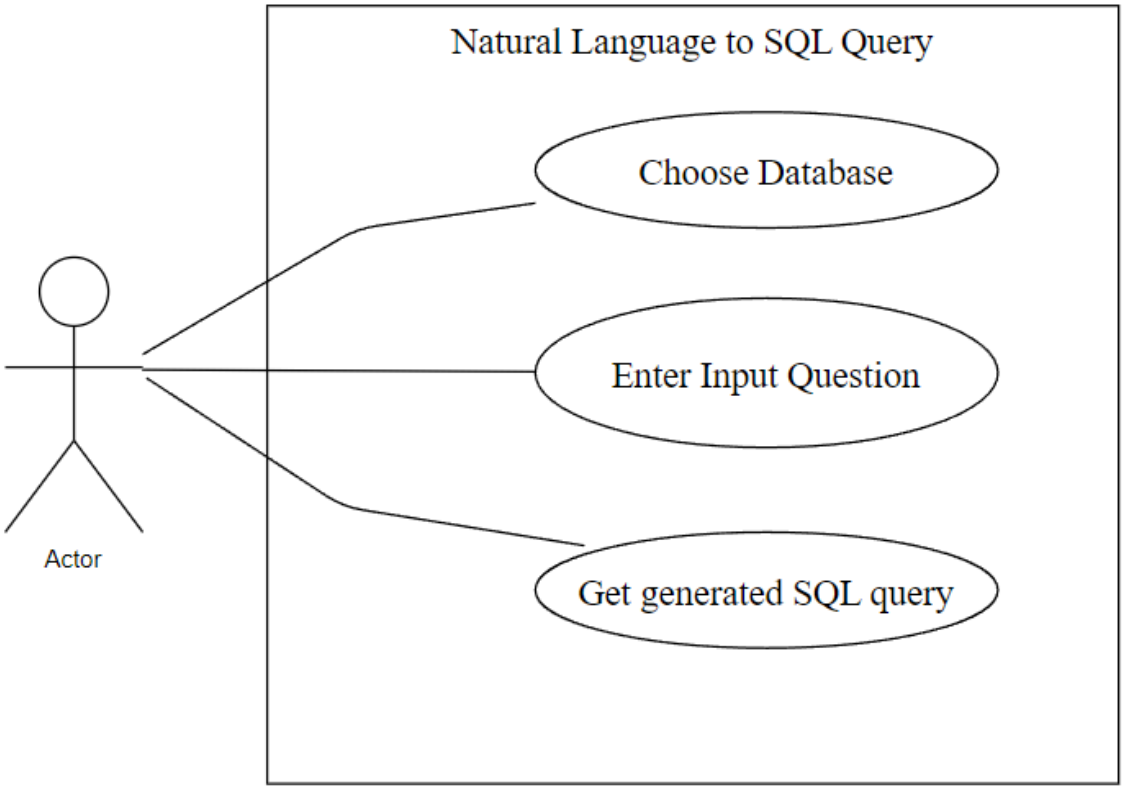

Figure 3.2.: Use Case Diagram

Use Case	Choose Database
Description	The user should be able to choose a database from the list of databases
Precondition	valid schemas of databases
Post Condition	The chosen database
Flow of Events	**Primary Flow** ● The user choose database from checkbox

Use Case	Input Question
Description	The user should be able to type the requested question
Precondition	chosen database
Post Condition	The question and the database are sent to the server
Flow of Events	**Primary Flow** ● The user enters the desired question ● The user clicks on Translate to SQL button **Error Flow** ● The user types in a language other than English ● The user types nothing

Use Case	Get generated SQL query
Description	The user should be able to get the generated query
Precondition	The Model finishes its processing with input
Post Condition	-
Flow of Events	**Primary Flow** ● The user gets the generated SQL query **Error Flow** ● If The user typed in a language other than English or he typed nothing he will get an UnTranslatable Error

3.2.2 Class Diagram

3.2.3 Sequence Diagram

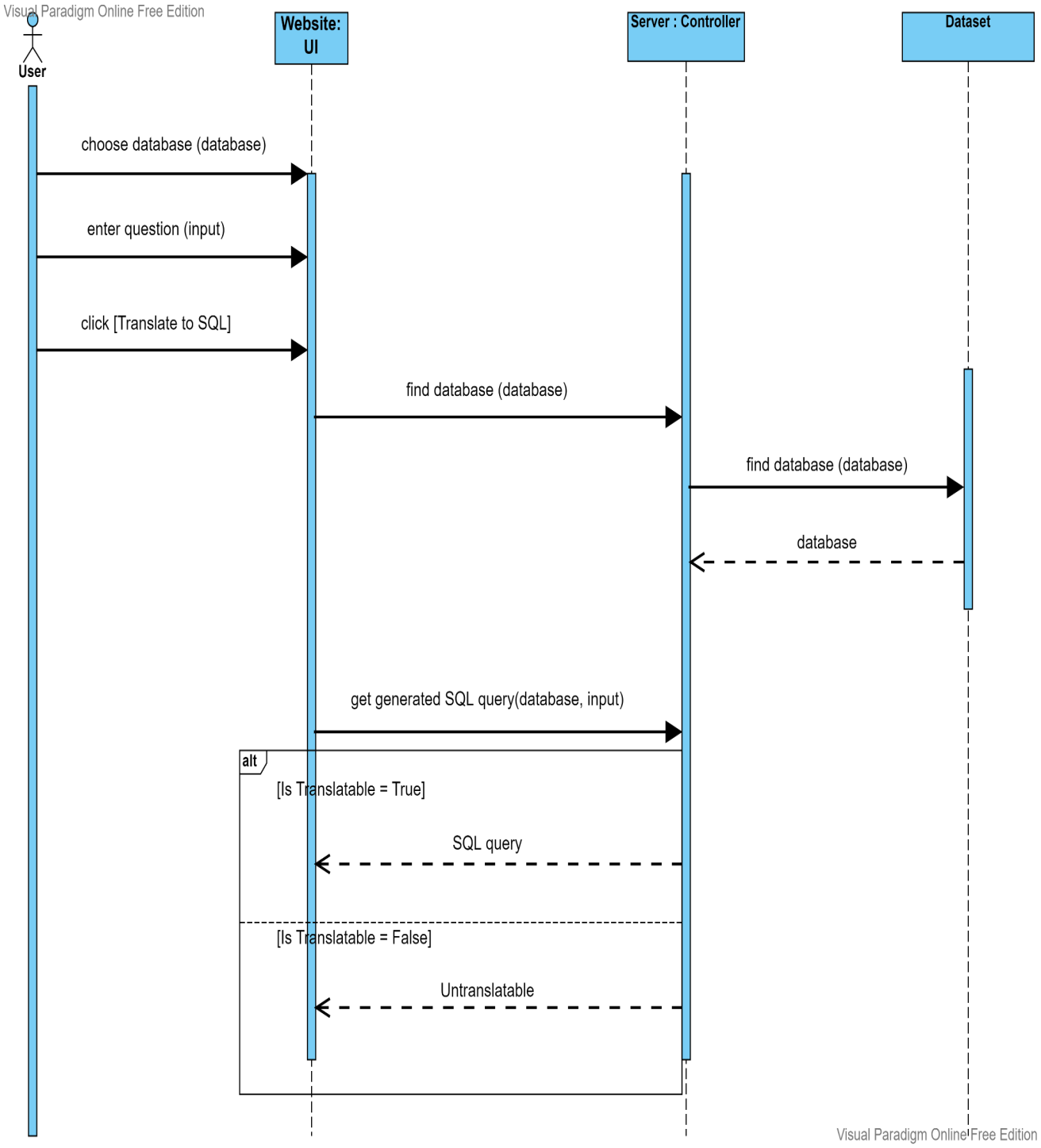

Figure 3.3.: Sequence Diagram

Implementation and Testing

In this chapter, we will explain the application's different modules and the role of each of them in the project.

Our modules are developed using Python, which is an interpreted, object-oriented, high-level programming language with dynamic semantics.

Specially used for web development as a back-end tool and machine learning applications.

The system enables the user to enter a question in English language and to choose the database of the domain that the question belongs to, then our model translates this statement to a SQL statement which could be used by the end user to retrieve any data from the databases related to his field.

4.1 The flow of the model:

- **Question Input**: The user inputs a question in Natural language and chooses a database name which is the most relatable to his question.

- **Translation:** the generated sequence (hybrid of the schema and the input) is passed to our translation model which outputs raw program sequences with probability scores using a beam search algorithm.

  To create the translation model, we used **Pytorch** machine learning framework which is a free and open-source software framework based on a library called **Torch**.

  It is used for applications related to Artificial intelligence such as Computer vision and Natural language processing.

- **Post Processing :** the raw program sequences are passed through a SQL checker, which verifies its syntactical correctness where sequences that failed to pass the checker are discarded from the output.

4.2 Dataset :

- **Spider dataset :**

Spider is a large-scale *complex and cross-domain* semantic parsing and text-to-SQL dataset annotated by 11 Yale students. The goal of the Spider challenge is to develop natural language interfaces to cross-domain databases. It consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables covering 138 different domains.

The preprocessing also included data cleaning and completing missing meta-data.

4.2 Model Architecture:

Main model network consists of the following components :

[1] Question-Schema encoding :

As shown in the following figure, we represent each table with its table name followed by its fields, before each table name we write symbol [T] to identify table names in the statement and symbol [C] to identify field names.

In case of a question that requires multiple tables (join queries), they are concatenated to form a serialization of the schema. The start and the end of tables and fields names are surrounded by the symbol [SEP] and the question is preceded by the symbol [CLS] to form the hybrid question - schema sequence.

$$X = [\text{CLS}], Q, [\text{SEP}], [\text{T}], t_1, [\text{C}], c_{11} \ldots, c_{1|T}$$
$$[\text{T}], t_2, [\text{C}], c_{21}, \ldots, [\text{C}], c_{N|T_N|}, [\text{SEP}].$$

Finally, The hybrid Sequence X (Shown in the previous figure) is passed to be encoded through **BERT,** Followed by bi-directional **LSTM.**

[2] Bridging :

Modeling only the table/field names and their relations is not always enough to capture the semantics of the schema and its dependencies with the question.

The solution to this problem is the **Fuzzy string matching** algorithm.

We perform a fuzzy string match between the question and the picklist of each field in the database. The matched field values are inserted into the

question-schema representation X, succeeding the corresponding field names and separated by the special token [V]. If multiple values were matched for one field, we concatenate all of them in matching order. If a question is matched with values in multiple fields. We add all matches and let the model (**BERT**) learn to resolve ambiguity.

[3] Decoder :

An LSTM with multihead attention is used as a decoder, where the decoder is initiated with the final state of the encoder.

At each step, the decoder performs one of the following actions :

- Generating a token from the vocabulary.
- Copying a token from the question.
- Copying a schema component from our current schema.

Mathematically, at each step t, given the decoder state st and the encoder representation [hQ; hS] ∈ R (|Q|+|S|)×n , we compute the multi-head attention through the following equations :

$$e_{tj}^{(h)} = \frac{\boldsymbol{s}_t W_U^{(h)} (\boldsymbol{h}_j W_V^{(h)})^\top}{\sqrt{n/H}}; \quad \alpha_{tj}^{(h)} = \operatorname*{softmax}_j \left\{ e_{tj}^{(h)} \right\}$$

$$\boldsymbol{z}_t^{(h)} = \sum_{j=1}^{|Q|+|S|} \alpha_{tj}^{(h)} (\boldsymbol{h}_j W_V^{(h)}); \quad \boldsymbol{z}_t = [\boldsymbol{z}_t^{(1)}; \cdots ; \boldsymbol{z}_t^{(H)}],$$

where h ∈ [1, . . . , H] is the head number and H is the total number of heads.

The probability of generating from V and the output distribution is defined as the following equation :

$$p^t_{\text{gen}} = \text{sigmoid}(\boldsymbol{s}_t \boldsymbol{W}^s_{\text{gen}} + \boldsymbol{z}_t \boldsymbol{W}^z_{\text{gen}} + \boldsymbol{b}_{\text{gen}})$$

$$p^t_{\text{out}} = p^t_{\text{gen}} P_{\mathcal{V}}(y_t) + (1 - p^t_{\text{gen}}) \sum_{j:\tilde{X}_j = y_t} \alpha^{(H)}_{tj},$$

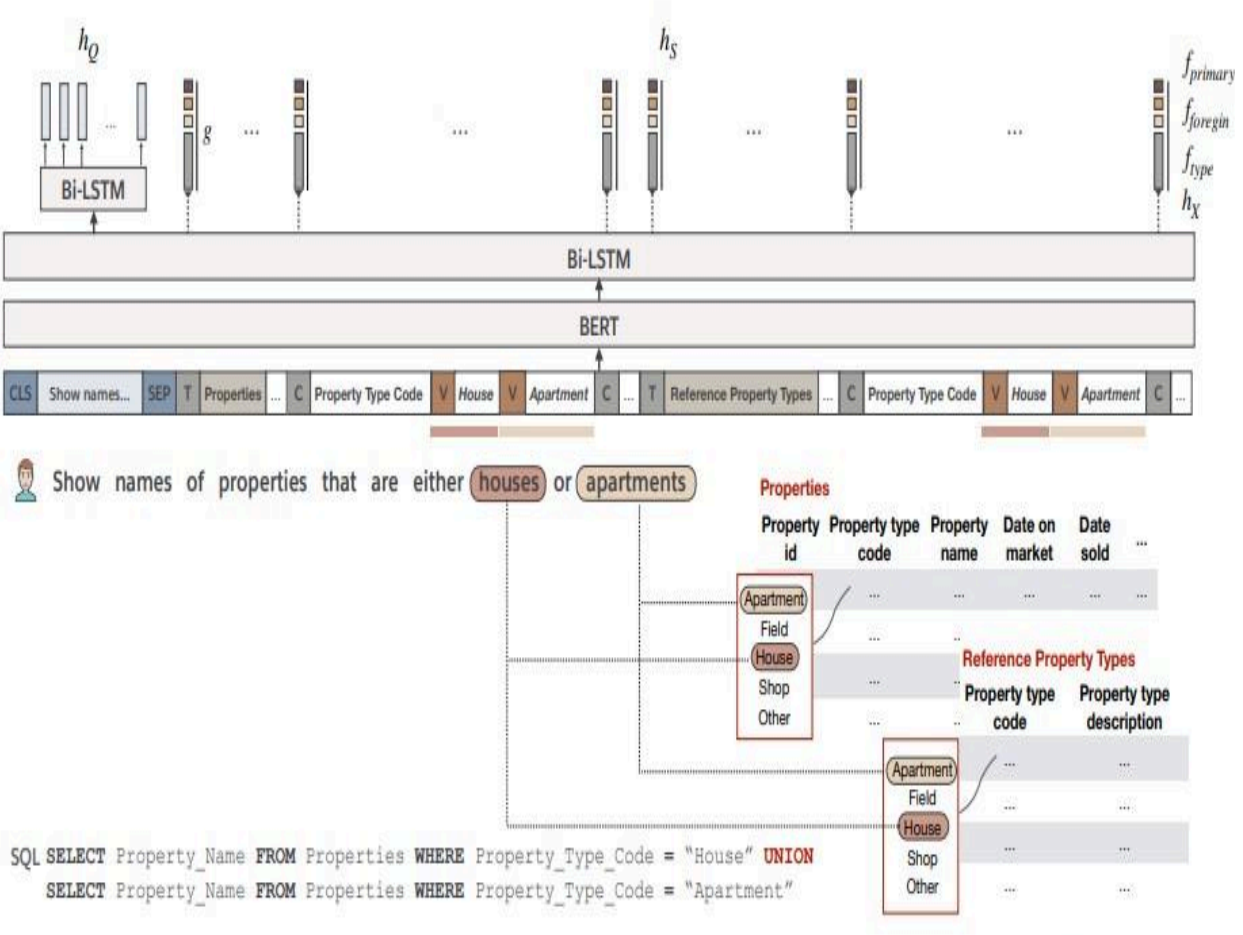

Figure 4.1.: Model Architecture

4.3 Training:

- The number of examples used for the training was 8695 examples trained for 25
epochs and mini batch size of 32.

- We train our model used the **BERT large** model and **LSTM** with 8-head attention
between the encoder and the decoder.

- We use **cross-entropy** as a loss function along with **Adam**-SGD with default parameters as an optimizer.

- LSTM hidden layer dimension is set to 400.

- **Learning rate** is set to 0.0005.

- We fine tune **BERT** model with a fine-tuning rate linearly increasing from 0.00003
to 0.00006. and shrinks at the end of the training process to 0.

4.4 Results :

- To validate the translation model, we use the **Exact Match (EM)** metrics which is
widely used in Question answering and machine translation tasks.

- The **Exact Match (EM)** metrics evaluates the structural correctness of the predicted
SQL by checking the orderless set match of each SQL clause in the predicted query.

- The best accuracies are shown in the following figure :

  - Top-1 exact match : 0.695.

  - Top-2 exact math : 0.741.

  - Top-3 exact match : 0.765

  - Top-5 exact match : 0.784

  - Top-10 exact match : 0.800

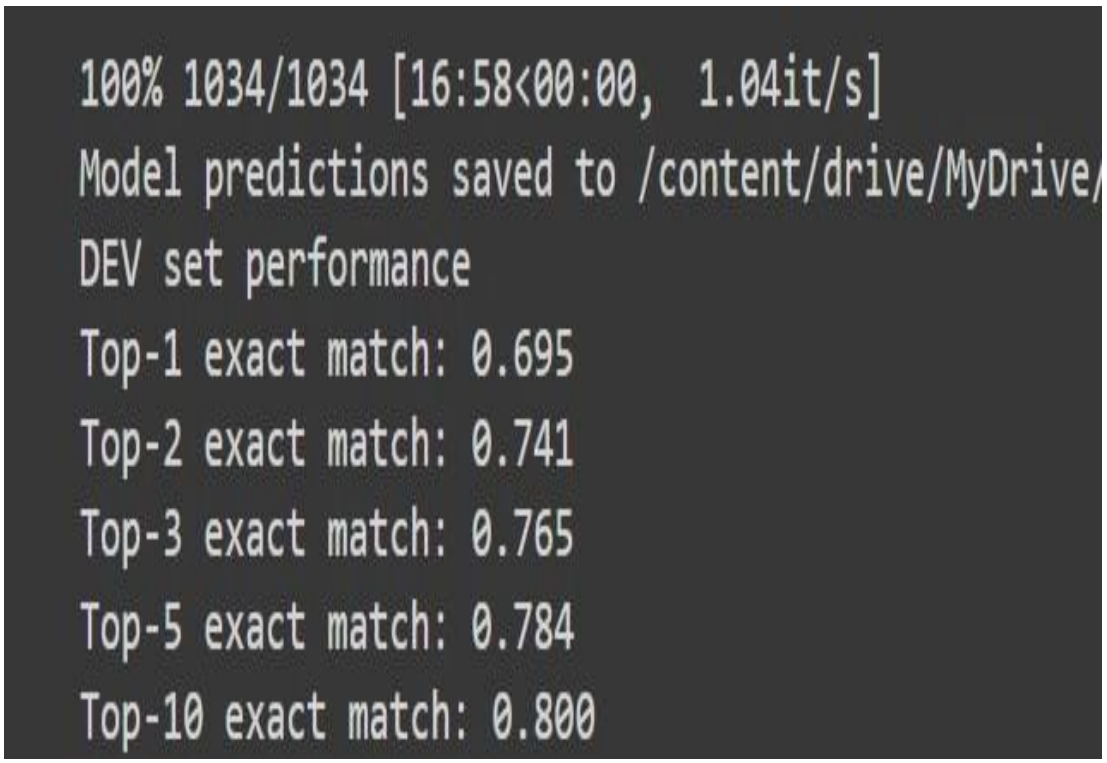

Figure 4.2. Results

4.5 Used environments :

- Our project is built as a web application using the following environments :

(1) Google colaboratory :

Colaboratory is a Google research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud.
We used google colab notebooks for our trails and models training. It provides an environment for developing and training models with the following specifications:

- 3.9 GB RAM

- 78.5 GB HARD DISK

(2) PyCharm :

PyCharm is an IDE used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.

(3) NLTK :

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

(4) Pandas :

Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the Numpy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables.

(6) PyTorch :

PyTorch is an open source machine learning framework for Python programs
that facilitates building deep learning projects.

4.6 Used technologies :

(1) **HTML**, css and bootstrap for the front-end part.

(2) **Flask** :

- Flask is a web framework for Python, meaning that it provides functionality for building web applications, including managing HTTP requests and rendering templates.

(3) **Ngrok** :

- A global distributed service for deploying web applications running in any
cloud, private network or local host.

5- User Manual

5.1 Overview :

- Our product is a web application where its main purpose is to query different database domains without prior knowledge of any programming language, This is done by translating the question of user asked in natural language (English language)
to SQL-form one, then this query could be used to get answers to the user's question.

5.2 Operating the web application :

5.2.1 Start page :

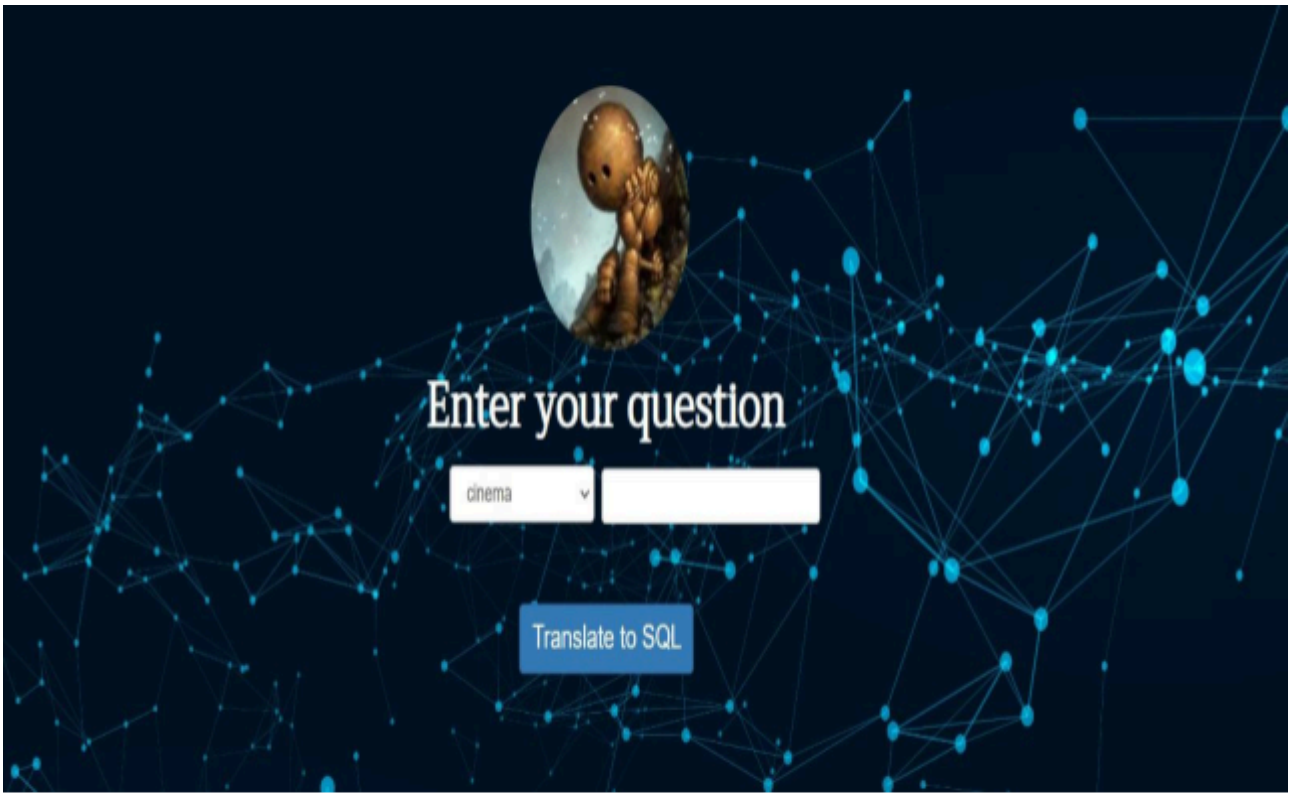

Figure 5.1. Start Page

5.2.2 choose the database related to the domain of your question :

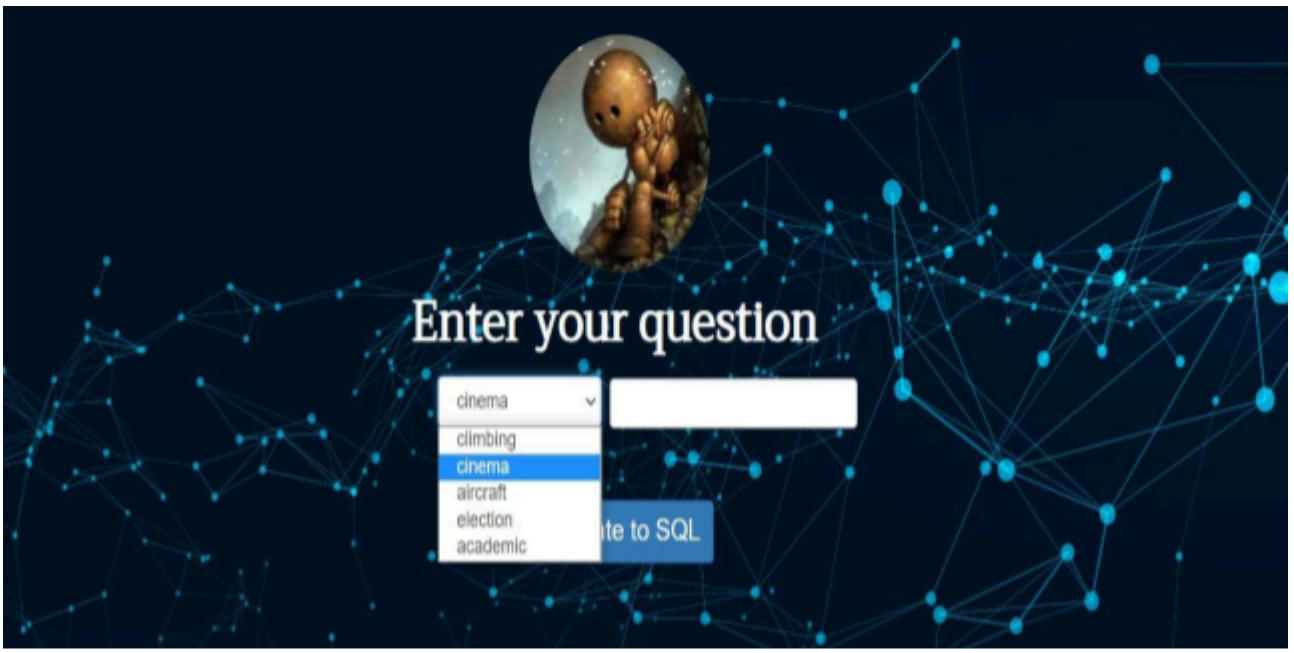

Figure 5.2. Choose DB

5.2.3 Enter your question in Natural language :

Figure 5.3. Enter Question

5.2.4 Click on “Translate to sql” button to start translation :

Figure 5.3. Translate to SQL

5.2.5 : Translation to SQL query is done :

```
SELECT film.Film_ID FROM film
```

Figure 5.4. Translation To SQL Done

6- Conclusion and Future Work

6.1 Conclusion

We present a powerful sequential architecture for modeling dependencies between natural language question and relational DBs in cross-DB semantic parsing. Our Model serializes the question and DB schema into a tagged sequence and maximally utilizes pre-trained LMs such as BERT to capture the linking between text mentions and the DB schema components. It uses anchor texts to further improve the alignment between the two crossmodal inputs. Combined with a simple sequential decoder with schema-consistency driven search space pruning, Our Model attained state-of-the-art performance on the widely used Spider text-to-SQL benchmarks.Our analysis shows that our model is effective at generalizing over natural language variations and memorizing structural patterns. It significantly outperforms previous work in the easy category of Spider. However, it struggles in compositional generalization and sometimes makes unexplainable mistakes. This indicates that when data is ample and the target logic form is shallow, sequence-to sequence models are good choices for cross-DB semantic parsing, especially given the implementation is easier and decoding is efficient. For solving the general text-to-SQL problem and moving towards production, we plan to further improve compositional generalization and interpretability of the model. We also plan to study the application of our model and its extensions to other tasks that require joint textual and tabular understanding such as weakly supervised semantic parsing and fact checking.

6.2 Future Work

- We plan to enhance our system to be language independent so it can support multiple languages (E.g : Arabic) where everyone around the world can benefit from our application.
- Also planning to support voice commands where users can easily request their information.

References

[1] Lin, Xi Victoria, Richard Socher, and Caiming Xiong. "Bridging textual and tabular data for cross-domain text-to-sql semantic parsing." *arXiv preprint arXiv:2012.12627* (2020).

[2] salesforce/TabularSemanticParsing: Translating natural language questions to a structured query language (github.com)

[3] Spider: Yale Semantic Parsing and Text-to-SQL Challenge (yale-lily.github.io)

[4] mozilla/moz-sql-parser: DEPRECATED - Let's make a SQL parser so we can provide a familiar interface to non-sql datastores! (github.com)

[5] Bazaga, Adrián, Nupur Gunwant, and Gos Micklem. "Translating synthetic natural language to database queries with a polyglot deep learning framework." *Scientific Reports* 11.1 (2021): 1-11.

[6] Yeo, Hangu. "A machine learning based natural language question and answering system for healthcare data search using complex queries." *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018.

[7] Xu, Boyan, et al. "NADAQ: natural language database querying based on deep learning." *IEEE Access* 7 (2019): 35012-35017.

[8] Ning, Zenan, et al. "Review of question answering technology based on Text to SQL." *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*. IEEE, 2021.

[9] Wang, Ping, Tian Shi, and Chandan K. Reddy. "Text-to-SQL generation for question answering on electronic medical records." *Proceedings of The Web Conference 2020*. 2020.

[10] Ahkouk, Karam, and Mustapha Machkour. "Towards an interface for translating natural language questions to SQL: a conceptual framework from a systematic review." *Int. J. Reason. based Intell. Syst.* 12.4 (2020): 264-275.

[11] Yeo H. A machine learning based natural language question and answering system or healthcare data search using complex queries. In2018 IEEE International Conference on Big Data (Big Data) 2018 Dec 10 .WikiSQL Dataset | Papers With Code

[12] Vaswani, Ashish, et al. "Attention is all you need."*Advances in neural information processing systems* 30 (2017).

[13] Welcome To Colaboratory - Colaboratory (google.com)

[14] ngrok - Online in One Line

[15] Kaliyar, Rohit Kumar. "A multi-layer bidirectional transformer encoder for pre-trained word embedding : a survey of bert." *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, 2020.

[16] Palasundram, Kulothunkan, et al. "SEQ2SEQ++ A Multitasking-Based Seq2seq Model to Generate Meaningful and Relevant Answers " *IEEE Access* 9 (2021): 164949-164975.

[17] Ma, Zeyu, et al. "Network Traffic Prediction based on Seq2seq Model." *2021 16th International Conference on Computer Science & Education (ICCSE)*. IEEE, 2021.

[18] Saini, Sandeep, and Vineet Sahula."A survey of machine translation techniques and systems for Indian languages." *2015 IEEE International Conference on Computational Intelligence & Communication Technology. IEEE, 2015.*

[19] Lauzon, Francis Quintal. "An introduction to deep learning." *2012 11th International Conference on Information Science.*