**PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA**

**MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH**

**MOHAMED BOUDIAF UNIVERSITY - M'SILA**

**FACULTY: Mathematics and Computer Science**

**DEPARTMENT: Computer Science**

**N°: …….**

**DOMAIN: Mathematics and Computer Science**

**FIELD: Computer Science**

**OPTION: Information Systems and Software Engineering**

**A Dissertation Submitted in Partial Fulfilment of the**

**Requirements for the Degree of Academic Master**

**By**:

Seghir Birem, Othman

## SUBJECT

# Generating SQL Queries from Spoken Natural Language

**Board of Examiners:**

| | | |
|---|---|---|
| Dr. Abdelbaki BOUGUERRA | University of M'sila | President |
| Dr. Mahmoud BRAHIMI | University of M'sila | Supervisor |
| Dr. Khadidja DERDOUR | University of M'sila | Examiner |

**Academic year: 2023 /2024**

## Dedications

***"What makes one's experiences really memorable is his loved ones who shares the happiness with him, and stands by in the sorrow… which makes me unhelpful to only be grateful as much as I can be for having such loving beautiful souls around me.***

***Naming the dedication of this humble work has to start with my ever-loving mother and my father who taught me how to be the man I am today, who kept standing by my side whenever my shoulders were weakened, whatever words I can think of and however I would put them together could never mount to what you both deserve. My siblings also are nothing less than to be mentioned in this everlasting text, thanks to them I get to be a happier man every single day.***

***At last, but not at all least, what is a man without his crew? What is life without friends? So special thanks and dedication to all of my brothers out there, DAGANG for life boys.***

***May ALLAH keep you all away from harm and safe by my side at all times."***

***Seghir Birem Othman***

## Acknowledgments

I first and foremost should and want to address whatever this work shall have of success as a gift solely from ALLAH, I couldn’t have achieved anything without his indispensable blessings and mercy, EL HAMDU-LI-ALLAH.

Special thanks go to my friends and family who have contained the hectic moments and stress that I have been through during the course of the research project, I am nothing but so blessed to have had the emotional support that I needed.

I am also deeply indebted to my thesis supervisor Brahimi Mahmoud whose steadfast support and inspirations have made this project a great success. In a very special way, I thank him for every support he has rendered unto me to see that I succeed in this challenging study.

I thank the faculty for giving me the grand opportunity to work in this interesting and knowledge-enriching project.

# Table of content

## List of Figures



## List of Tables

# GENERAL INTRODUCTION

In the contemporary digital era, the exponential growth of digital information has reached unprecedented levels, compelling organizations to handle vast and intricate datasets daily. This surge necessitates sophisticated, easy-to-use, and effective tools for data management and analysis. Structured Query Language (SQL) remains a robust choice for interacting with large relational databases, offering the capability to seamlessly retrieve, manipulate, and organize extensive datasets. However, SQL's rigidity and complexity can pose challenges, particularly for non-technical users. Pre-programmed queries, which are common in many systems, often fail to address the dynamic and evolving needs of users, limiting their ability to extract diverse and meaningful insights from the data. This project addresses these challenges by exploring alternative approaches that balance SQL's efficiency with the flexibility needed in today's dynamic data environments.

The primary objective of this project, is the design and implementation of a model named SQUIRE, which stands for "SQL Queries Using Intelligent and Robust Engine." SQUIRE is envisioned as an intuitive Natural Language Database Interface (NLDBI) for SQL relational databases, aimed at generating relevant SQL queries without requiring extensive SQL knowledge from users. This model seeks to democratize data access, making it easier for a broader range of stakeholders to leverage data for informed decision-making. Additionally, SQUIRE enables users to create varied queries that are not pre-programmed in the existing system, offering greater flexibility and adaptability in data retrieval.

To achieve this, the project commenced with an extensive study of Natural Language Processing (NLP), a key branch of Artificial Intelligence. This included reviewing its primary applications, techniques, and methodologies to identify the most suitable frameworks for our model. Following this foundational study, we conducted a preliminary investigation into the core problems associated with SQL and its components. This phase involved the design of our system, detailing the adopted architecture and the algorithms and pipelines integrated within it.

Once we had the system's architecture in place, we moved on to the implementation phase. We chose the right tools and frameworks to develop SQUIRE. During this phase, we also prepared and processed the datasets needed to train the model. We created an effective training strategy and made sure the model performed well by carefully testing and debugging it. Finally, we analyzed the model's performance using advanced metrics, assessed its usability and efficiency, and identified areas for future improvements.

To achieve the aforementioned objectives, this manuscript is structured as follows:

**Chapter One: Natural language processing** – This chapter explores Natural Language Processing (NLP) and its relevant techniques, providing a foundation for the subsequent design of the model.

**Chapter Two: Design of the Model** – This chapter presents the conceptualization and structure of SQUIRE, presenting the adopted architecture and the algorithms and pipelines used.

**Chapter Three: Implementation** – This chapter details the practical aspects of building and refining the model, including the selection of frameworks, preparation of datasets, training strategies, and performance evaluation.

Finally, the manuscript concludes with a comprehensive summary of the findings and offers suggestions for future research and improvements.

# CHAPTER I: NATURAL LANGUAGE PROCESSING

## 1 Introduction

Natural Language Processing (NLP) is a fascinating field in the intersection of computer science, linguistics, and artificial intelligence. It is primarily concerned with giving computers the ability to manipulate and interpret human language in a way that is meaningful and useful to us. Just as if we communicate with each other using words, sentences, and conversations, NLP aims to enable computers to do the same thing with humans and with each other.

As NLP technology continues to advance, it is expected to see applications that are even more exciting and more advanced. From better language translation tools to more intuitive virtual assistants, NLP has the potential to revolutionize how humans interact with technology and each other.

In this chapter, we will delve deeper into this paradigm, exploring its applications, associated methodologies, and technological frameworks to provide a comprehensive understanding.

## 2 NLP: Definition and basic Concepts

In this section, we will delve into the essence of this field, elucidating its fundamental concepts in detail.

### 2.1 Definition

Natural language processing (NLP) refers to the branch of computer science and more specifically the branch of artificial intelligence (AI), concerned with giving computers the ability to understand text and spoken words in much the same way human beings can. [1]

NLP combines computational linguistics, rule-based modeling of human language with statistical machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment. [1]

NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly even in real time. It also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes. [1]

## 2.2 Evolution and Historical development of NLP

Natural language processing has its roots in the 1950s. Already in 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is called currently the Turing test as a criterion of intelligence, though at the time that was not articulated as a problem separate from artificial intelligence. The proposed test includes a task that involves the automated interpretation and generation of natural language.

### 2.2.1 Symbolic NLP (1950s – early 1990s)

John Searle's Chinese room experiment effectively summarizes the concept of symbolic NLP: Given a set of rules (such as a Chinese phrasebook with questions and corresponding answers), the computer simulates natural language understanding (or other NLP tasks) by applying those rules to the data it encounters. [31]

The 1980s and early 1990s were the heyday of symbolic approaches in NLP. The focus areas at the time included research on rule-based parsing (e.g., the development of HPSG as a computational operationalization of generative grammar), morphology (e.g., two-level morphology), semantics (e.g., Lesk algorithm), reference (e.g., within Centering Theory), and other areas of natural language understanding (e.g., in Rhetorical Structure Theory). Other areas of study were pursued, such as the creation of chatterbots using Racter and Jabberwacky.

The increasing relevance of quantitative evaluation during this time period was a significant development (which eventually led to the statistical shift in the 1990s). [31]

### 2.2.2 Statistical NLP (1990s – 2010s)

Prior to the 1980s, most natural language processing systems were built on elaborate sets of handwritten rules. However, beginning in the late 1980s, a revolution in natural language processing occurred with the development of machine learning techniques. This was owing to both the constant development in computational capacity and the gradual decline in the supremacy of Chomskyan theories of linguistics, whose theoretical foundations prohibited the type of corpus linguistics that underpins the machine-learning approach to language processing. [31]

Since the mid-1990s, as the web has grown, more raw (unannotated) language data has become available. As a result, unsupervised and semi-supervised learning algorithms have received more attention in research. Such algorithms can learn from data that hasn't been manually annotated with the required responses, or from a combination of annotated and unannotated data. In general, this activity is far more challenging than supervised learning and gives less accurate outcomes for the same quantity of input data. However, there is a tremendous amount of unannotated data available (including, among other things, the whole content of the World Wide Web), which can sometimes compensate for the poor results provided the method used has a low enough time complexity to be useful. [31]

### 2.2.3 Neural NLP (Present)

In 2003, word n-gram model, at the time the best statistical algorithm, was over-performed by a multi-layer perceptron (with a single hidden layer and context length of several words trained on up to 14 million of words with a CPU cluster in language modelling) by Yoshua Bengio with co-authors. [31]

In 2010, Tomáš Mikolov (then a PhD student at Brno University of Technology) and co-authors applied a simple recurrent neural network with a single hidden layer to language modeling. He later developed Word2vec. In the 2010s, representation learning and deep neural network-style (with many hidden layers) machine learning methods gained popularity in natural language processing. This popularity was fueled in part by a flurry of results demonstrating that such techniques can achieve cutting-edge performance in a wide range of natural language tasks, including language modeling and parsing. This is increasingly important in medicine and healthcare, where NLP helps analyze notes and text in electronic health records that would otherwise be inaccessible for study in order to improve care or protect patients. [31]

In 2023, Gautam Siwach (a PhD student at the University of New Haven) and co-author Dr. Cheryl Li present an innovative approach to improving Human-Cobot Interaction (HCI) by incorporating Natural Language Processing (NLP). The research incorporates the CoboVox voice recognition system into the UR3e cobot interface, allowing for seamless voice-activated engagement and efficient communication. The methodology entails creating a comprehensive vocabulary, evaluating NLP models, and selecting the best one for practical application. The resulting HCI framework makes voice commands easier to understand, allowing even inexperienced robot operators to program and control cobots with ease. This study represents a significant advancement in the field, highlighting NLP's transformative role in bridging the communication gap between humans and collaborative robots. [31]

# 3 Fields of application

As we delve into the expansive realm of NLP, it becomes clear that its impact extends far beyond mere linguistic analysis. NLP's versatility and sophistication enable its integration into a variety of fields, each benefiting from its unique capabilities. From healthcare and finance to marketing and customer service, those are some of the most famous ones:

## 3.1 Information Retrieval (IR) and Information Extraction (IE)

### 3.1.1 Information Retrieval (IR)

Information retrieval encapsulates all operations associated to organizing, processing, and accessing information in all forms and formats. An information retrieval system enables users to interact with an information system or service in order to locate information - text, graphic images, sound recordings, or video - that meets their specific requirements.

Thus, the goal of an information retrieval system is to help users retrieve important information in an orderly collection of documents. In reality, most information retrieval systems are actually document retrieval systems, as they are intended to retrieve information about the presence (or non-existence) of documents relevant to a user query. [2]

### 3.1.2 Information Extraction (IE)

It is the process of scanning text for information relevant to a certain interest, including the extraction of entities, relations, and, most challengingly, events--or who did what to whom when and where. It necessitates more in-depth analysis than key word searches, but its goals fall short of the extremely difficult and long-term task of text comprehension, in which we want to capture all of the information in a text, as well as the speaker's or writer's intention.

Information extraction is somewhere in the middle of this spectrum, with the goal of capturing structured information while maintaining practicality. IE typically concentrates on surface language phenomena that do not need deep inference, as well as the phenomena that appear most frequently in texts. [3]

## 3.2 Virtual assistants and conversational agents (chatbots)

Virtual assistants, also known as intelligent conversational systems such as Google's Virtual Assistant and Apple's Siri, interact with human-like responses to users' queries and finish specific tasks. These virtual assistants are termed as "dialogue systems often endowed with human-like behavior", and they have started becoming integral parts of people's lives. They did not just allow us to talk to computers via commands but to accomplish our day-to-day tasks also, by using natural human language. [4]

## 3.3 Automatic Translation

Also known as, Machine Translation refers to the process of translating text or speech from one language to another using computational algorithms and without human intervention. These algorithms analyze the input text or speech and generate a corresponding translation in the target language. Automatic translation systems vary in complexity, from simple rule-based approaches to more sophisticated neural network models. These systems increasingly advanced over the years, driven by developments in artificial intelligence and neural language processing. Automatic translation has various applications, including multilingual communication, content localization, and cross-lingual information retrieval. [5]

### 3.4 Sentiment Analysis

Sentiment Analysis (SA) or Opinion Mining (OM) is the computational study of people's opinions, attitudes and emotions toward an entity. The entity can represent individuals, events or topics, which are most likely to be covered by reviews. SA can be considered as a classification process with three main classification levels. The first of which is document-level SA that aims to classify an opinion document as expressing a positive or a negative opinion or sentiment, and Sentence-level SA that aims to classify sentiment expressed in each sentence by determining whether it expresses a positive or a negative sentiment in case where the sentence is subjective. Lastly, there is aspect-level SA that aims to classify the sentiment with respect to the specific aspects of entities, by first identifying the entities and their different aspects. [6]

### 3.5 Automatic summarization

Automatic summarization, also known as text summarization, is the process of condensing a longer document or set of documents into a more concise version while retaining key information and main points. This process is typically carried out using computational algorithms that analyze the content of the text and select the most relevant sentences or passages for inclusion in the summary. Automatic summarization can be extractive, which selects sentences or passages directly from the original text, or abstractive, which generates new sentences to convey the main ideas in a more concise manner. This technique is widely used in information retrieval, document summarization, and text analysis applications. [7]

### 3.6 Text Classification

Text classification, also known as text categorization or document classification is the task of assigning predefined categories or labels to natural language text documents, based on their content. It is a fundamental problem in information retrieval and natural language processing, with applications ranging from document organization and management to email filtering and sentiment analysis. [8]

## 3.7 Natural Language Interfaces (NLIs)

A Natural Language Interface is a system that interprets and executes natural language commands, as opposed to commands in a specialized command language. The goal of a natural language interface is to facilitate communication between users and a computer system in a manner that resembles human-to-human communication as closely as possible. [9]

### 3.7.1 Natural Language Interfaces to Database (NLIDBs)

Are specialized subsets of NLIs, which enable users to access information stored in a database by formulating Natural Language Queries (NLQs). By seamlessly integrating natural language processing capabilities with database querying functionality, NLIDBs streamline the interaction between users and databases, making data retrieval more intuitive and accessible. This further reinforces the objective of NLIs to enhance communication between humans and computer systems by leveraging the natural language proficiency of users. [10]

### 3.7.2 Intelligent Database Systems (IDBSs)

An IDBS is equipped with a data management system capable of managing enormous amounts of persistent data, from which various forms of reasoning can be applied to derive new data and information. This involves knowledge representation approaches, inference techniques, and intelligent user interfaces that go beyond the typical query language approach by utilizing natural language facilities. These strategies play a significant role in improving database systems. Knowledge representation techniques enable improved representation of the semantics of application domains in the database, inference techniques enable reasoning about data to extract additional data and information, and intelligent user interfaces assist users in making requests and receiving responses. [11]

### 3.8 Healthcare and Medicine

It is difficult for electronic healthcare systems to understand the information contents of the unstructured and narrative texts from the diverse offered tools such as the Electronic Medical Records (EMR) or Electronic Health Records (EHR). Simply because they are composed of heterogeneous grammatical structures, varied expressions expressed in a variety of natural languages as well as the use of diverse terms to denote particular concepts. Consequently, the healthcare domain is characterized by ambiguity and also by high costs and high error rates. Nevertheless, NLP techniques have been used to structure information from narrative texts so as to provide data for decision making [12]. Alternatively, as well as:

- Improving clinical documentation: Rather than wasting valuable time manually reviewing EHR, NLP uses speech-to-text dictation and formulated-data entry to extract critical data from EHR at the point of care.
- Accelerating clinical trial matching: Using NLP, healthcare users can automatically review vast quantities of unstructured clinical and patient data and identify eligible candidates of clinical trials.
- Supporting clinical decisions: NLP makes fast, easy and efficient for physicians to access health related information exactly when they need it, enabling them to make decisions that are more informed whenever they need it.

## 4 NLP techniques and phases

### 4.1 Tokenization

Tokenization is the process of separating input text into components, known as tokens, to be processed by computers. These tokens are subsequently used in further natural language processing stages such as morphological analysis, word-class labeling, and parsing. Tokenization commonly involves identifying both sentence and token boundaries, as future treatments typically target particular sentences [13]. Furthermore, the process of tokenization can be done on multiple levels and algorithms depending on the complexity of the intended problem, the given data, or the available computing power, from which we can mention the following:

### 4.1.1 Word tokenization

Which involves breaking down text into meaningful parts such as words, or symbols, by taking natural breaks from pauses in textual data splitting them into tokens using delimiters. Word tokenization is typically characterized with less precise tokens but less vocabulary sizes since every word as a whole are given a specific encoding to represent it.

### 4.1.2 Character tokenization

In this kind of tokenization each word is broken into the characters used to spell it, this shall eliminate problems that occurs when the model is missing some words in its vocabulary. Although drastically increasing the size of the input and output of the model. Character level tokenizers are typically known to have very little vocabulary sizes since tokens represent only characters, although these are also characterized with extensive sequence sizes due to the fact that every single character is a token on its own.

### 4.1.3 Sub-word tokenization

This process of tokenization involves breaking words into smaller chunks to better help understand their meaning like prefixes or suffixes that clarifies the word's function. This makes it easier for models to understand words outside of their vocabulary since it can break down even the words outside of the vocabulary, although one downside this method can have is large vocabulary sizes which results sometimes in intensive computation necessity. The most used methods to achieve sub-word tokenization are Byte Pair Encoding (BPE) [32], or word-piece [35].

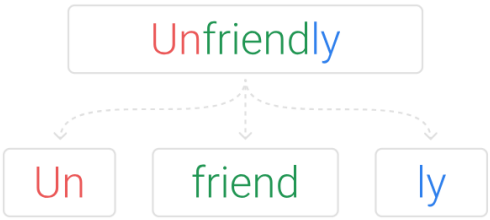


*Figure 1 Sub-Word tokenization*

This particular type of tokenization provides very precise tokens due to the fact that it splits each word to the parts that gives it meaning (e.g. Prefixes and suffixes), which makes it the best-chosen tokenization technique for NLP tasks that requires a deep understanding of the given corpus. Sub-word tokenizers also have moderate sized vocabulary depending on the training corpora and its complexity.

### 4.2 Part-of-Speech tagging

Part-of-speech (POS) tagging, sometimes termed grammatical tagging, is the automatic attribution of part-of-speech tags to words in a sentence, a POS is a grammatical classification that generally comprises verbs, adjective, adverbs, nouns etc. POS tagging is a significant NLP application for machine translation, word sense disambiguation, question answering parsing, and so on. The origins of POS tagging can be traced back to the uncertainty of many words' parts of speech in context. [14]

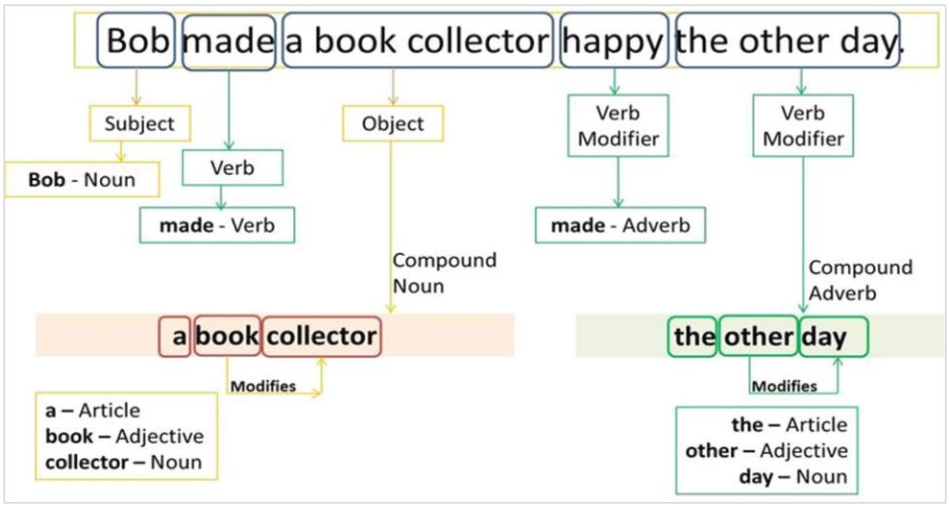


*Figure 2 POS tagging*

### 4.3 Lemmatization

Lemmatization combines vocabulary and morphological analysis to remove inflectional endings from words, returning them to their dictionary form. It ensures that things are done correctly by assessing whether query terms are used as verbs or nouns. It also assists in matching synonyms using a thesaurus, so that when searching for "ids" the word "children" is matched as well. [15]

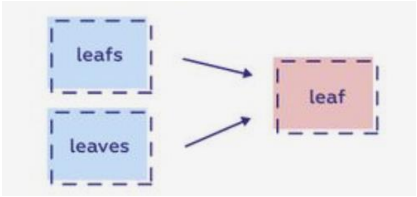


*Figure 3 Lemmatization*

### 4.4 Stemming

Stemming is a technique used in information retrieval systems to ensure that variants of words are not excluded when text is recovered. The procedure is used to remove derivational suffixes and inflections (suffixes that affect the shape of a word and its grammatical function), allowing word variants to be merged into the same roots or stems. [15]

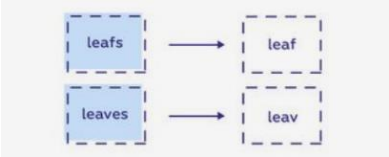


*Figure 4 Stemming*

### 4.5 Named entity recognition

Named Entity recognition (NER) involves analyzing a text and determining which instances of words or expressions fall into which Named Entity (NE) categories. For activities like information extraction, information retrieval, and other text processing applications, NE recognition software is a crucial preprocessing tool. [16]

### 4.6 Parsing

Natural language parsing challenges entail finding a sentence's syntactic structure (parse tree), which describes its grammatical organization. There are two forms of parsing which are dependency parsing and constituency parsing. Dependency parse trees are formed on direct relationships between words or other tokens in a phrase, whereas constituency parse trees are based on formal grammar's parse trees.

#### 4.6.1 Dependency Parsing

Dependency parsing is a syntactic analysis approach in natural language processing that seeks to discover a sentence's grammatical structure by finding word associations (dependencies). Each word in the sentence is represented as a node in a graph, with dependencies between words represented as directed edges that indicate syntactic connections such as subject, object, modifier, and so on. [17]

#### 4.6.2 Constituency Parsing

Constituency parsing is a syntactic analysis approach in natural language processing that divides sentences into grammatical components or phrases. Unlike dependency parsing, which focuses on the links between words in a phrase, constituency parsing seeks to determine a sentence's hierarchical structure using a preset grammar, often expressed as a parse tree. Constituency parsing decomposes sentences into nested phrases, such as noun phrases, verb phrases, and prepositional phrases, using syntactic rules provided by the language. [18]

### 4.7 Semantic Analysis

Semantic Analysis examines the grammatical structure of sentences, including the arrangement of words, phrases, and clauses, to establish links between independent elements while accounting for context, logical structuring of sentences and grammar roles. It is also an essential component of many machine learning systems accessible today, including search engines, chatbots, and text analysis software. [19]

### 4.8 Relationship Extraction (RE)

Relation extraction is a natural language processing (NLP) job that identifies and extracts meaningful relationships between things stated in text. These associations can contain terms such as "is_Married_to," "works_for," "located_in," and any other link between entities. Relation extraction aims to automatically discover and categorize the connections between things stated in text, allowing machines to better grasp the meaning and context of the information given in textual data. [20]

### 4.9 Probabilistic and Neural Language Models (NLMs)

#### 4.9.1 Probabilistic Language Models

The probabilistic language model aims to compute a probability distribution of a sentence of words, or sequences of words, i.e. $P(s) = P(w_1, w_2, \dots, w_N)$, or to compute the probability of an upcoming word. Namely, given a sentence, $s = (w_1, w_2, \dots, w_N)$, where wt are discrete words, N is the random-valued sequence length, and the goal is to compute the probability of an upcoming word $P = (w_t | w_1, \dots, w_{t-1})$. In many natural language-related applications, estimating the probabilistic language model is particularly helpful. For machine translation, language modeling can aid in word ordering and selection. [21]

### 4.9.2 Neural Language Models

A neural language model (NLM) is made to forecast the probability distribution of a word sequence in a natural language. In order to identify and comprehend intricate patterns and connections in the linguistic data, neural network designs are employed. Through extensive training on vast text datasets, these models approximate the probability of particular word sequences in light of prior context, facilitating applications like sentiment analysis, machine translation, and language generation. Because NLMs naturally cluster related words together and project words into low-dimensional space, they perform better than typical probabilistic models. [21]

## 4.10 Word Embeddings

Word embeddings are representations of words. The embedding is used in text analysis. Typically, the representation is a real-valued vector that encodes the meaning of the word in such a manner that adjacent words in the vector space are assumed to have a similar meaning. Language modeling and feature learning approaches may be used to produce word embeddings, which include mapping words or phrases from the vocabulary to vectors of real numbers. [22]

## 4.11 Attention Mechanism

An attention mechanism is a computational technique that allows models to focus on certain bits of input data (such as words or phrases) when making predictions or producing results. Attention processes, which originated in the field of neural machine translation, have become crucial in a variety of NLP applications. They improve model performance by allowing them to dynamically balance the value of distinct input pieces during processing, effectively capturing long-range dependencies and enhancing context handling. This attention mechanism is often implemented using neural network designs and has greatly aided the growth of NLP tasks such as machine translation, text summarization, and question answering. [23]

# 5 NLP's Approaches and solutions

## 5.1 Term frequency – inverse document frequency (TF-IDF)

TF-IDF (word Frequency-Inverse Document Frequency) is a statistical metric used in information retrieval to determine the importance of a word inside a document in comparison to a corpus of documents. This metric is calculated by multiplying the term frequency (TF), which measures the frequency of a word's occurrence inside a document, by the inverse document frequency (IDF), which reduces the weight of terms that appear often across the corpus. TF-IDF seeks to highlight phrases that are common inside a text but less common across the corpus, hence identifying terms of relevance relevant to the content of a document. TF-IDF, which is widely used in a variety of applications such as document categorization, information retrieval, and text mining, is an important tool for extracting meaningful insights from text. [25]

Given a document collection D, a word *w*, and an individual document d ϵ D, we calculate:

$$w_d = f_{w,d} \times \log(\frac{|D|}{f_{w,D}})$$ [25]

## 5.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network that models sequential data while keeping an internal state or memory. Unlike feedforward neural networks, which flow data in a single direction from input to output, RNNs feature connections that create directed cycles, allowing them to display dynamic temporal activity. This cyclic structure allows RNNs to handle input sequences of varied lengths while still capturing relationships between components in the sequence. [26]

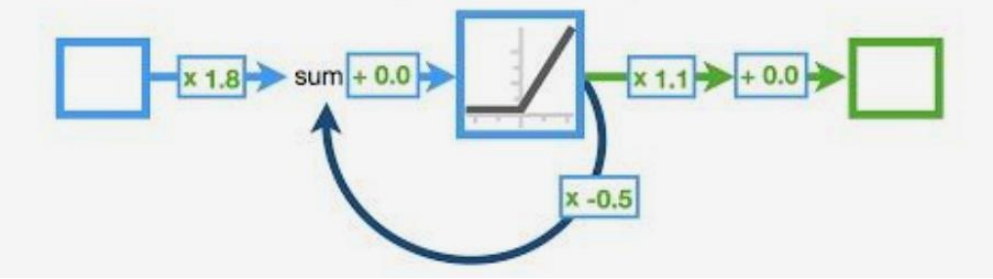


*Figure 5 Recurrent neural network*

In RNNs, the hidden state at each time step acts as a memory, storing information from past time steps and influencing the calculation at the current time step. This recurrent connection enables RNNs to demonstrate "memory" or "contextual understanding," making them ideal for applications requiring sequential input, such as language modeling, time series prediction, and sequence synthesis. [26]

Traditional RNNs are good at modeling sequential data, but they frequently suffer from the disappearing or expanding gradient problem, which restricts their capacity to capture long-range relationships. This dilemma has led to the creation of different RNN versions, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which overcame these concerns and have become generally accepted in practice. [26]

### 5.3 Long short-term memory (LSTM)

Hochreiter and Schmidhuber, first developed long Short-Term Memory, or LSTM, in 1997. It is a type of recurrent neural network architecture that is used in audio and picture identification, as well as natural language processing, and was specifically designed to address the vanishing and exploding gradient problems that traditional RNNs face when dealing with long sequences. LSTM networks have specialized memory cells that can retain information over extended time periods, allowing them to successfully capture long-term dependencies in sequential data. [27]

At each time step, an LSTM unit takes an input, updates its internal state depending on the current input and the information stored in the memory cell, and provides an output as well as a new memory cell state. The major innovation of LSTM units is their gating mechanisms, which include input, forget, and output gates that regulate the flow of information into and out of the memory cell. These gates allow LSTM networks to selectively keep or reject input across time, which aids in the learning of complicated temporal patterns. [27]

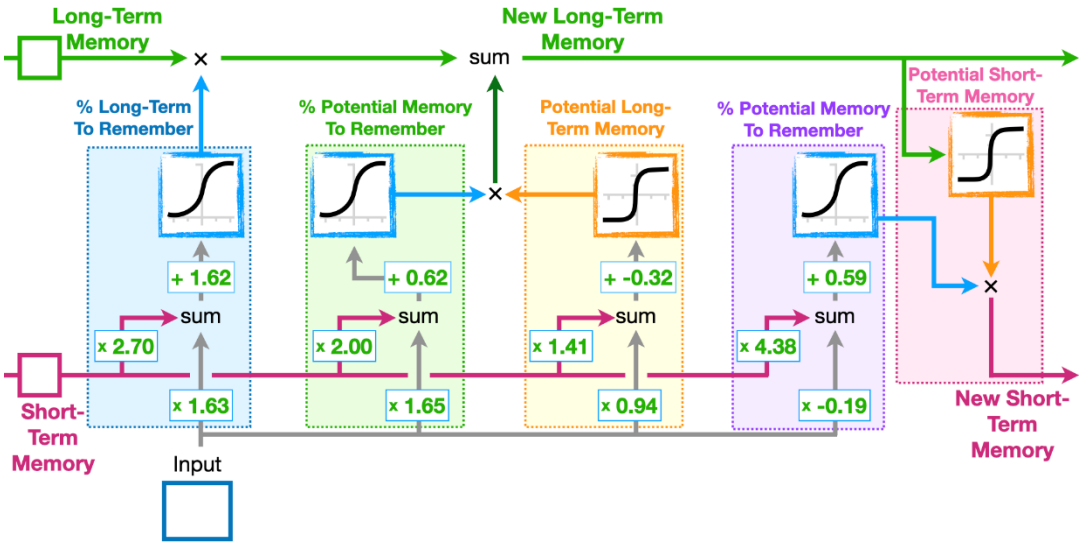


*Figure 6 Long Short-term memory*

### 5.4 Gated Recurrent Unit (GRU)

GRUs are a type of recurrent neural networks, specifically designed to address the vanishing gradient problem in traditional recurrent neural networks. Proposed by Cho et al. in 2014 to make each recurrent unit capture the dependencies of different time scales adaptively. Similarly, the GRU like the LSTM has gating units that modulate the flow of information inside the unit, however without having any separate memory cells, enabling better long-term dependencies modeling. These gating mechanisms allow GRUs to selectively update and forget information, making them particularly effective for tasks involving sequential data, such as natural language processing and time series prediction. [28]

### 5.5 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a type of feedforward neural network that is specifically intended to analyze organized grid-like input, such as images. Unlike classic fully connected neural networks, CNNs take advantage of the spatial structure in the input data. They do this by using convolutional layers, with each layer including learnable filters (also known as kernels) that slide over the input data, extracting local patterns and features. [26]

CNNs consist of three main components: convolutional layers, pooling layers, and fully connected layers. Convolutional layers conduct the fundamental function of convolving input data with learnable filters, resulting in feature maps that record hierarchical representations of the input. Pooling layers lower the dimensionality of feature maps by combining local information, making the network more resistant to input translations and distortions. Fully linked layers use the spatially localized characteristics learnt by previous layers to produce predictions or classifications. [26]

## 5.6 Supervised and Unsupervised Training

Machine learning involves the development of algorithms and models that allow computers to learn from data and make predictions or decisions without explicit programming. Machine learning is fundamentally based on the notion of model learning, which involves algorithms improving their performance on a job via experience. There are two main types of machine learning techniques: supervised learning and unsupervised learning.

### 5.6.1 Supervised Learning

Supervised learning is a type of machine learning in which the algorithm learns from labeled dataset. Each input in the dataset has a matching output, which provides examples for the algorithm to use in order to learn. During this process, the algorithm generalizes from the labeled data and produces predictions about unseen cases. Regression and classification are two common techniques in supervised learning. Regression predicts continuous outcomes, whereas classification assigns categorical labels to inputs. [29]

### 5.6.2 Unsupervised Learning

Unsupervised learning is a branch of machine learning in which algorithms learn from unlabeled data. Unsupervised learning has no specified outputs, and the algorithm aims to identify hidden patterns or structures in the data. Clustering algorithms, such as k-means, group similar data points together, whereas dimensionality reduction techniques, such as principal component analysis (PCA), seeks to represent data in a lower-dimensional space while retaining its important properties. [30]

## 5.7 Transformer Models

A transformer is a deep learning architecture created by Google that uses the multi-head attention technique. Text is translated into numerical representations known as tokens, which are then turned into vectors by looking up from a word embedding database. At each layer, each token is contextualized into the context window with other (unmasked) tokens using a concurrent multi-head attention method, allowing the signal for crucial tokens to be increased while less important tokens are decreased. [24]

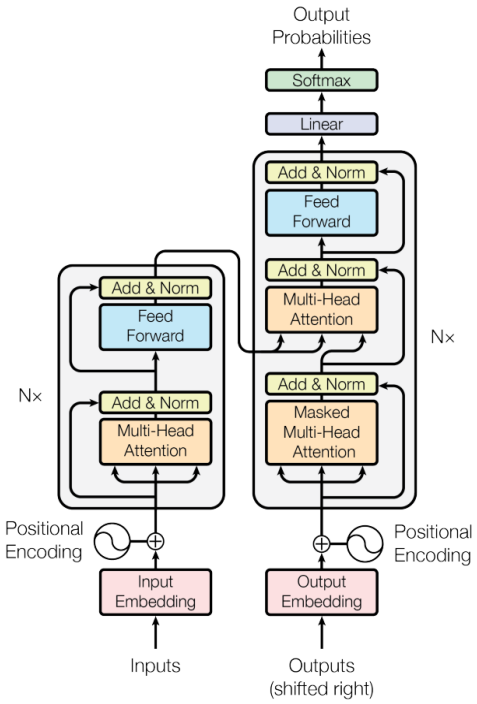


*Figure 7 Transformer model architecture*

Transformers can be implemented in multiple architectures depending on the use case and the intended task, the main architectures are:

### 5.7.1 Encoder-Decoder transformers

These models use both the encoder and decoder components of the transformer architecture. The encoder processes the input and produces a context vector to pass as an input for the decoder, which then generates the output sequence.

### 5.7.2 Encoder-only transformers

They consist only of the decoder part of the transformer architecture, and mainly used in the tasks of processing input sequences in order to generate contextual embeddings, or in another words, understanding the input.

### 5.7.3 Decoder-only transformers

Which consist only of the decoder part of the transformer architecture, they can generate text in a unidirectional manner, so they can predict the next word in a sequence based on the previous context, which enables these models to be used in text generation, language modeling, and dialogue systems. Which makes these architectures ideal for chatbots.

## 6 Conclusion

In this chapter, we looked at Natural Language Processing (NLP) and its many applications in a variety of fields. From information retrieval to sentiment analysis, NLP has surely transformed the way we engage with technology and information in our day-to-day life.

We also explored important NLP techniques and phases, including tokenization and neural language models such as Transformers. In addition to a variety of methodologies, including traditional methods like TF-IDF and modern deep learning systems like RNNs and CNNs.

Now, as we approach implementation, we are ready to put our knowledge into action. In the following chapter, we will create an NLP model for transforming natural language queries into SQL queries.

# CHAPTER II: THE MODEL DESIGN

## 1 Introduction

In the field of human-machine interaction, exploiting the intuitive potential of natural language is an evolutional task. Unlike structured computer languages, natural language has an inbuilt ease of expression, making it an ideal mediator between humans and machines, hence, it can be a very fitting alternative of SQL language for those who does not master it. This chapter takes on a tour through the intricate design process of our model, SQUIRE, acknowledging its role as a bridge that intuitively combines human intents and machine comprehension, with the use of Natural Language Processing techniques and approaches.

The following chapter shall demonstrate every single step taken in the process of designing SQUIRE with its according architectures, diagrams, and followed techniques to better-understand what goes around inside the system in order to produce the desired results.

## 2 SQL overview

### 2.1 Introduction to SQL

Structured Query Language (SQL) is a powerful, standardized programming language used to manage and manipulate relational databases. Since its debut in the 1970s, SQL has been the foundation of database management, allowing users to effectively query, update, insert, and delete data within a database. Its simple syntax and powerful capabilities enable the creation and management of database structures, assuring data integrity and security. SQL is widely used across a variety of database systems, including MySQL, PostgreSQL, Oracle, SQLite, and Microsoft SQL Server, making it a necessary expertise for database administrators, developers, and data analysts. Individuals who master SQL can use relational databases to successfully organize, analyze, and exploit data in a variety of applications, ranging from web development to corporate intelligence.

## 2.2 SQL components

SQL is composed of several sub-languages, each serving a distinct purpose in the management and manipulation of data within relational databases. The primary sub-languages of SQL are Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL) and more [37], which we will delve into in the following elements:

### 2.2.1 Data Definition Language (DDL)

DDL's use case is to define and manage the structure of database objects such as tables, indexes, and schemas. It includes commands that create, alter, and drop these objects, thus shaping the database schema. Key DDL commands include CREATE, ALTER, DROP, and TRUNCATE. [37]

### 2.2.2 Data Manipulation Language (DML)

DML's prime usage is to manipulate data stored within database tables. It includes commands that insert, update, delete, and query data. Key DML commands include SELECT, INSERT, UPDATE, and DELETE. [37]

### 2.2.3 Data Control Language (DCL)

DCL is used to control access to data within the database. It includes commands that grant and revoke permissions and manage user access. Key DCL commands include GRANT, and REVOKE. [37]

### 2.2.4 Transaction Control Language (TCL)

TCL is a database transaction-management language that ensures data integrity and consistency. It contains directives that govern the transactional behavior of a database activity. Key TCL commands include COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT, and SET TRANSACTION. [37]

### 2.2.5 Data Query Language (DQL)

Although sometimes considered a subset of DML, DQL focuses specifically on querying the database to retrieve data. The primary DQL command is SELECT, which is used to query the database and retrieve data from one or more tables. [37]

# 3 The importance of converting NL commands into SQL Queries

## 3.1 Detailed problem definition

In today's digital world, the exponential growth of digital information and its importance has peaked in comparison with all past decades. As a result, organizations are dealing with vast and intricate datasets on a daily basis in order to manage and fully utilize the market's potential. Although, the sheer volume and complexity of data brings with it unique growing challenges, including the need for sophisticated, easy-to-use, and effective tools for data handling, management and analysis, in order to reach all stakeholders. Furthermore, SQL in this context stands as an indomitable force. However, as we delve deeper into the intricacies of data handling, it becomes apparent that while SQL provides a robust framework, its rigidity and complexity in certain scenarios can pose challenges. This particular trade-off prompts a closer examination of alternative approaches and solutions that can nail a balance between the efficiency of SQL and the adaptability demanded by today's diverse and evolving data environments.

## 3.2 Proposed solution and project purpose

Natural language database interfaces (NLDBIs) emerge as the most fitting solution for addressing the challenges by SQL's rigidity and complexity of use. NLDBIs bridge the gap between users and databases by allowing them to interact in natural language, removing the need for users to have expertise in SQL syntax. This not only enhances accessibility but also empowers a wider range of stakeholders within enterprises to query and analyze data without relying on specialized technical knowledge. An NLDBI can leverage advanced NLP techniques in order to understand user commands, translating them into SQL queries, thereby combining the power of SQL with the intuitiveness of human language.

Hence, our set of goals in this project includes designing and implementing an easy-to-use NLDBI that can generate relative SQL queries, to the particular chosen database, with the best-known techniques and approaches, to achieve the best performance and accuracy possible.

# 4 System Architecture

The task of generating SQL queries from natural language commands is a special kind of translation tasks since the model should take a sequence and return another one as an output. Hence, the best models available to achieve this particular job are mainly seq2seq models [42], and transformer models [24]. Both models have an amazing ability to process sequences and understand their context although with different levels of accuracy and range.

Transformers use self-attention mechanisms [24], which allows them to process and relate all parts of an input sequence simultaneously. This capability enhances their ability to capture long-range dependencies and contextual relationships more effectively than seq2seq models, who are practically implemented using LSTMs [27], typically rely on recurrent layers and process sequences in a step-by-step manner. Consequently, transformers achieve superior performance in various natural language processing tasks.

To wrap things up, Seq2Seq models were one of the first neural network architectures used in NLP tasks. Although, their limitations have been addressed by the transformer architecture, which has become the most popular choice for many NLP tasks due to its ability to process long sequences of data and capture global dependencies, as well as the introduction of pre-trained models, which encouraged us to choose the transformer architecture as the main architecture for our model.

## 4.1 Process Pipeline

In transformer models, the input raw text has to go through an intricate chain of processing, which is the model's pipeline, in order to produce the desired sequence, or the SQL query in this particular case. Although, these processes can be customized depending on the target-desired task, after a careful study and a set of experiments that will be discussed further in this chapter, we built our pipeline architecture as shown in the following figure:

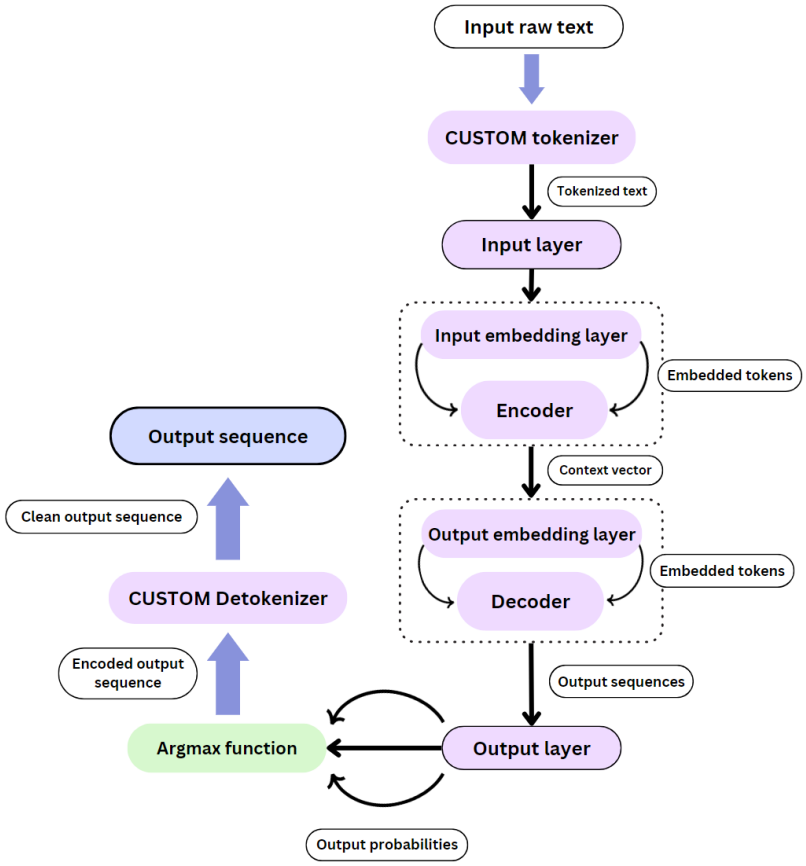


*Figure 8 Process pipeline*

In addition, since the desired task includes generating relative SQL queries, meaning that the queries generated are intended to be based on a specific chosen database by the user, we need to embed the database's schema with the input text so we can provide the transformer model with relative context information.

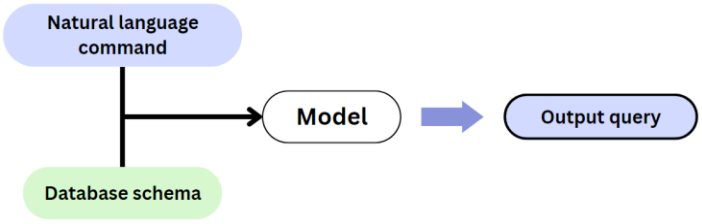


*Figure 9 Input sequence schema*

At last, when connecting a single database to the model, it would be computationally wasteful to pre-process the database's schema at every inference. So, the schema shall be pre-processed only at the first inference then concatenated to the later-preprocessed input command until the user changes the connected database, so changes the schema, that's when the new schema would be pre-processed again in order to use it in further inferences in order to use less resources.

# 5 Used Models and algorithms

## 5.1 Input and output layers

The input layer serves a fundamental role in receiving raw sequences of tokens and ensuring that these sequences conform to the model's requirements for maximum input and output sizes. These layers do not have trainable parameters; instead, their primary function is to manage the dimensions and structure of the input data ensuring uniformity across all of it.

Similarly, the output layer is meant to format the model's predictions into a consistent structure suitable for subsequent processing or evaluation. By standardizing input and output sizes, these layers facilitate efficient batch processing and ensure that the data is correctly prepared for the embedding layers and subsequent transformer architecture components, which performs the core computational tasks.

## 5.2 Custom Word-Piece Tokenizer/Detokenizer

Considering the fact that the model has to take two merged sequences of input, more precisely the database schema concatenated with the natural language command, we relatively have long sequences of input, which could result in the model having extensive sizes of input and consequently being excessively complex and harder to train. Furthermore, due to the specified characteristics of the task we can clearly see that sub-word tokenization is the best-suited thanks to its precise representation of tokens and moderate vocabulary sizes, and more relatively, shorter tokenized sequences.

The designed tokenizer is a custom one, trained specifically on natural language commands to perform advanced NLP tasks. Our tokenizer was instantiated using the Tokenizers library from HuggingFace [39], and employs the SentencePiece [33] library with the WordPiece algorithm [35], and trained on a middle-size dataset containing 400,000 text samples, with a vocabulary size of 15 thousand, ensuring robust and efficient tokenization capabilities.

The first process implemented in the pipeline of WordPiece tokenizers is normalization, and it can be implanted using a stack of normalizers, which could result in a more efficient text normalization. Hence, our normalization stack consists of two normalizers:

- StripAccent Normalizer: StripAccents is a normalization technique that removes diacritical marks (accents) from characters, meaning that it transforms accented characters into their unaccented equivalents. This process involves decomposing characters into their constituent parts and then removing the diacritical marks, for example, the character "é" is decomposed into "e" and the acute accent, and the accent is removed. [44]
- NFC Normalizer: NFC stands for Normalization Form C (Canonical Composition), it is one of the Unicode normalization forms, which standardize the way characters are represented in text. [43]

To improve performance, BERT's pre-tokenizer [41] was incorporated, allowing for more precise and robust sub-word tokenization. Additionally, the following set of special tokens has been utilized:

- “<s>”: beginning of sentence token.
“<pad>”: padding token.
“</s>”: end of sentence token.
“<unk>”: unknown token. Usually used to mark up an unknown token by the tokenizer.
“<cls>”: the classify token, which is typically represents the input sequence.
“<sep>”: separation token. Used to mark up a separation between two sequences.
“<mask>”: missing word token, typically used to handle the missing-word predictions.

These special tokens facilitate various tasks such as sequence padding, sentence separation, and masking for model training.

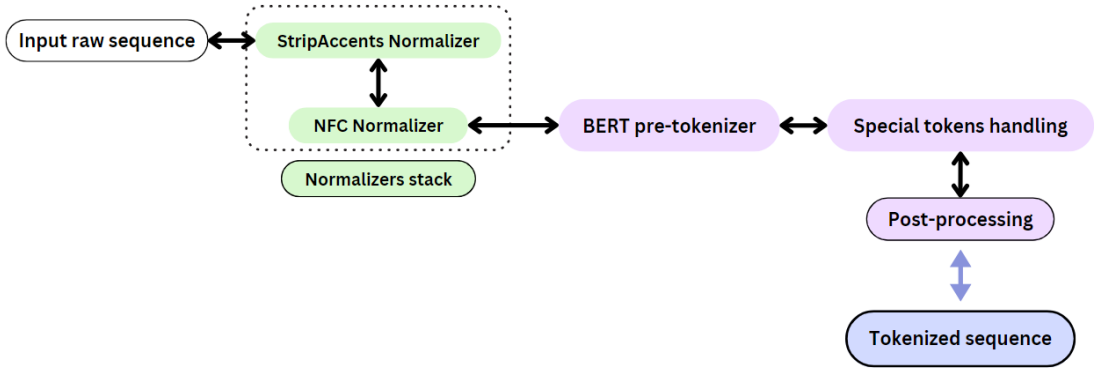


*Figure 10 Custom tokenizer architecture*

It is necessary to note that the demonstrated process is bi-directional in both cases of the tokenizer or the detokenizer, meaning that we can always provide an input sequence for the tokenizer and get a tokenized sequence as an output. The opposite direction would mean providing the encoded sequence and receive the textual one as an output.

The tokenizer then was wrapped in an object of the class PreTrainedTokenizerFast [40], to ensure seamless integration with diverse NLP frameworks. After training, the tokenizer has been saved, preserving its configuration and learned vocabulary for future use. This approach ensures that the tokenizer can handle tokenization tasks with high accuracy and performance, making it highly effective for downstream NLP applications.

The detokenizer in the other hand is the very same as the tokenizer with some slight key differences. Since these two components are meant to deal with two different types of input we trained the tokenizer on Natural Language commands only, and the detokenizer was trained only on SQL queries. Another key difference is the vocabulary size, meaning that the SQL vocabulary is very little compared to that in natural language, the detokenizer has a vocabulary size of only 5 thousand.

### 5.3 Embedding layer

In a transformer architecture, the embedding layer is responsible for mapping discrete input tokens (e.g., words, sub-words, or characters) to continuous vector spaces in order to pass them into the encoder/decoder blocks. This transformation is critical because it allows the subsequent neural network layers to operate on numerical data, facilitating the learning of complex patterns and relationships in the input sequences. The input to the transformer model typically consists of sequences of tokens, each represented by an index in a vocabulary of fixed size ($V$). [24]

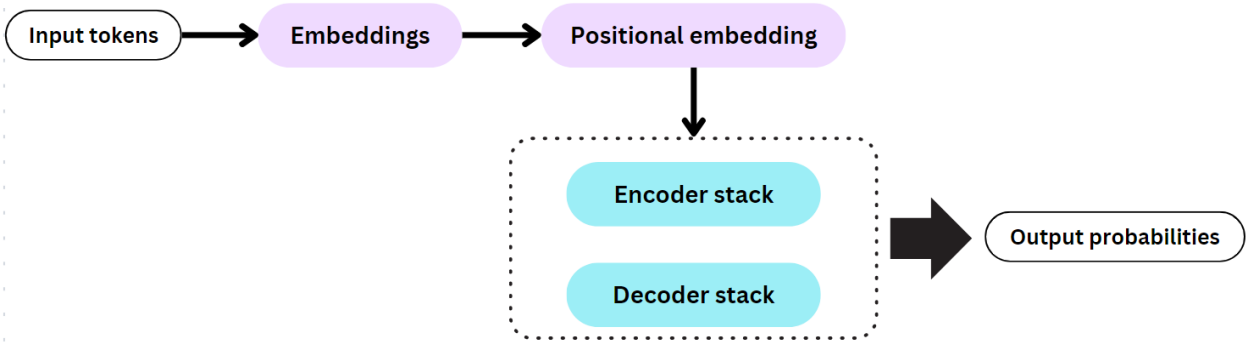


*Figure 11 Embedding layer architecture*

The input to a transformer is a sequence of tokens, $T = [t_1, t_2, \dots, t_n]$, where ti is the token index and n is the sequence length. Then comes the core of the embedding layer, which is a trainable matrix (E) of dimensions (V × d), where (d) is the dimensionality of the embeddings. Each row ($E_i$) in this matrix corresponds to the vector representation of the i-th token in the vocabulary. [24]

Hence, for any given input sequence the embedding layer retrieves the following vectors, $X = [E[t_1], E[t_2], \dots, E[t_n]]$, if for example $n$ = 30 and $d$ = 512, then $X$ is 30 × 512. [24]

Since the model does now contain any recurrence or convolution, it has to inject some information to make use of the order of the sequence. That's where the Positional Encoding PE(i) whom are added to the embeddings in order to incorporate tokens order comes handy:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}),\ PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$ [24]

Furthermore, given the sequence $T = [t_1, t_2, \dots, t_n]$ as the input, then the transformation will be as follows:

$$X = [E[t_1] + PE(1), E[t_2] + PE(2), \dots, E[t_n] + PE(n)],$$ [24]

### 5.4 Encoder-Decoder transformer model

Due to their ability to handle complex dependencies and contextual nuances in text, transformers are particularly best suited for Machine Translation (MT) and seq2seq [36] tasks. The attention mechanism at the core of transformers enables them to consider the relevance of each word in a sentence relative to every other word, capturing intricate relationships and meaning more effectively than traditional models [24]. This ability is indispensable for accurately interpreting user commands and translating them into precise SQL queries, which require an understanding of both the intent and the specific structure of the database language.

Furthermore, in the encoder-decoder architecture of the transformer models the encoder converts the input sequence into a rich, contextual representation called the context-vector, capturing intricate relationships and dependencies within input data. The decoder then takes that vector to generate the output sequence. This design has proven its high efficiency in tasks such as MT particularly. For instance, the transformer model's self-attention mechanism [24], which operates with a complexity of $O(n^2)$ per layer (where $n$ is the sequence length), allows it to capture long range dependencies more efficiently than RNN or LSTM networks. [24]

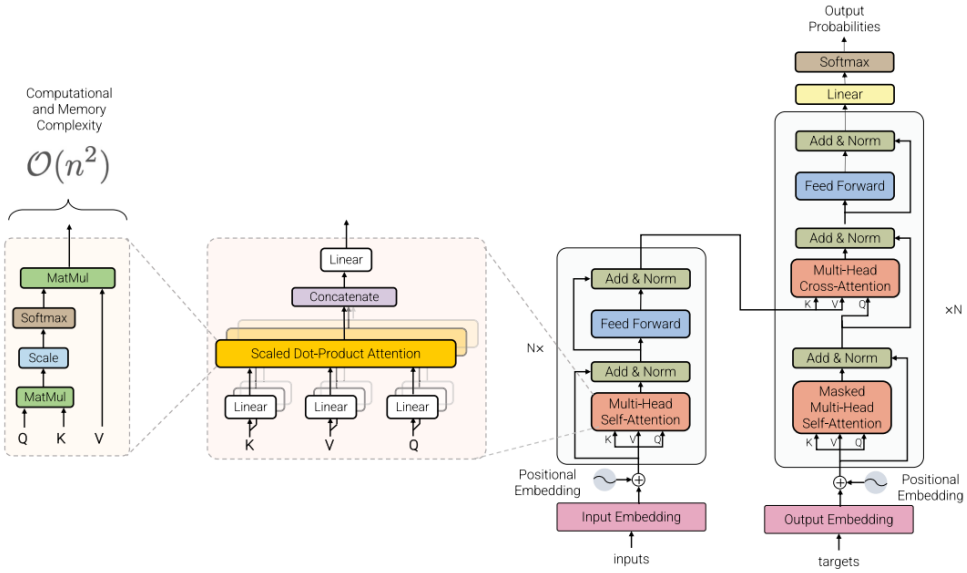


*Figure 12 Multi-head attention in Encoder-Decoder transformers*

A mapping between a query and a collection of key-value pairs and an output, where the query, keys, values, and output are all vectors, is how an attention function is defined. The results are calculated as a weighted sum of the values, with each value's weight determined by the query's compatibility function with its matching key. [24]

### 5.4.1 Scaled dot-product attention

Scaled dot-product attention is a method used to determine the influence of each input token on the representation of other tokens within the same sequence. It computes attention scores by taking the dot product of query and key vectors, scaling these scores, applying a soft-max function to obtain attention weights, and then using these weights to combine the value vectors. [24]

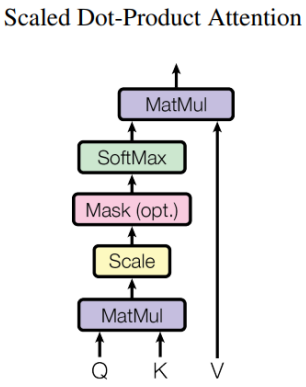


*Figure 13 Scaled Dot-Product attention*

In practice, the attention function is computed on a set of queries simultaneously, packed together into a matrix *Q*. The keys and values are packed also together into matrices *K* and *V* [24], the final attention output computation formula then is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \text{[24]}$$

### 5.4.2 Multi-head attention

Multi-head attention divides the attention mechanism into several "heads," each learning different aspects of the input data independently. The outputs from these heads are then concatenated and linearly transformed to produce the final output. This process allows the model to attend to information from different representation subspaces at different positions. [24]

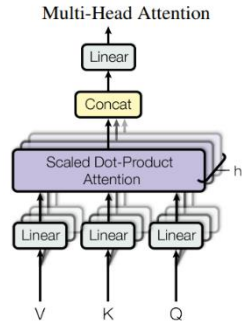


*Figure 14 Multi-head Attention*

It is more advantageous to linearly project the queries, keys, and values h times using distinct, learnt linear projections to $d_k$, and $d_v$ dimensions, respectively, rather than executing a single attention function with $d_{model} - dimensional\ keys$, values, and queries. The attention function is then performed in parallel to each of these projected copies of the queries, keys, and values to produce $d_v$-dimensional output values, these are concatenated and once again projected, resulting in the final values [24], as demonstrated in the figure above.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, \dots, head_n)WO, \text{ [24]}$$

$$\text{Where: } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V), \text{ [24]}$$

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

### 5.5 The Argmax function

The Argmax function is a fundamental concept in mathematics and is widely used in various fields for optimization and decision-making purposes. It plays a crucial role in finding optimal solutions and estimating parameters in statistical models. Furthermore, it is used in various fields such as machine learning for example, the argmax function is often used in algorithms like classification, where it is used to predict the class label with the highest probability. [38]

Given a function *f*(*x*), the argmax function returns the value of *x* that maximizes *f*(*x*), in mathematical notation it can represented as:

$$\boldsymbol{argmax_x}\, f(x) = \{x \mid f(x) \geq f(y), \forall y\}, \text{ [38]}$$

This means that $\boldsymbol{argmax_x}$ *f*(*x*) is the set of all *x* where *f*(*x*) is maximized. Although this last is the argmax of a discrete function, so the argmax can also be defined for a continuous function as follows:

$$\boldsymbol{argmax_x}\, f(x) = \{x \mid \frac{df(x)}{dx} = 0\ , \frac{d^2 f(x)}{dx^2} < 0\}, \text{ [38]}$$

In this case, the argmax corresponds to the point where the derivative of $(x)$ is zero and the second derivative is negative (indicating a local maximum).

## 6 Model's architecture

Careful consideration of hyper-parameters is an essential design phase to optimize performance, and key hyper-parameters include the dimensionality of the model layers ($d_{model}$), which defines the size of the embeddings and hidden layers, and the dimensionality of the feed-forward network ($d_{ff}$), which determines the capacity of the intermediate layers within each transformer block. There is also the number of heads, which encapsulates the number of attention heads in the multi-head attention mechanism. Every head can focus on different parts of the input sequence, allowing the model to capture various aspects of the data's context.

In addition, the dropout rate is applied to the outputs of each sub-layer. Dropout is a regularization technique used to prevent overfitting by randomly setting a fraction of activations to zero during training. Furthermore, the final hyper-parameters of the model are as follows:

- Number of heads: 4
- Number of layers: 2
- Dropout rate: 0.1
- Encoder input size (Input maximum size): 259
- Decoder input size (Output maximum size): 261
- Input vocabulary size: 15,000
- Output vocabulary size: 5000
- Dimensionality of the model's layers ($d_{model}$): 64
- Dimensionality of the feed-forward Network ($d_{ff}$): 256

After constructing the model relatively to the available data and compiling it, we can summarize the model's overall architecture as a whole in the following table:

| **Layer (type)** | **Output shape** | **Param #** | **Connected to** |
|---|---|---|---|
| Input_layer (**InputLayer**) | (None, 259) | 0 | - |
| Input_layer1 (**InputLayer)** | (None, 261) | 0 | - |
| encoder (**Encoder**) | (None, 259, 64) | 1,159,424 | input_layer[0][0] |
| decoder (**Decoder**) | (None, 261, 64) | 652,416 | input_layer1[0][0], encoder[0][0] |
| dense_1 (**Dense**) | (None, 261, 5000) | 325,000 | decoder[0][0] |

*Table 1 Model architecture summary*

- Total parameters: 2,136,040 (8.15 MB)
- Trainable parameters: 2,136,040 (8.15 MB)
- Non-trainable parameters: 0 (0.00 B)

The described architecture has been designed to be specifically lightweight and efficient, making it suitable for tasks where computational resources are limited or quick model iteration is needed. The use of four attention heads and two layers per encoder/decoder is a balanced choice for simpler or smaller-scale NLP tasks. However, in order to implement more-complex tasks, deeper architectures with larger dimensions might be required to achieve state-of-the-art performance, so due to the hardware limitations we are facing, the previous architecture is suitable to demonstrate the potential of the model at least.

## 7 Conclusion

In this chapter, we explored the design of our model for translating natural language into SQLite commands. We began with an overview of SQL, its components, and its significance in database management. We then addressed the importance of converting natural language into SQL, highlighting the benefits for users.

We detailed the model architecture, supported by diagrams, illustrating the process from natural language input to SQL generation. Each component was described, along with the algorithms and techniques used. This chapter provides a foundation for the implementation phase, which the next chapter shall uncover, transitioning from the theoretical design to practical application.

# CHAPTER III: The Model Implementation

## 1 Introduction

This chapter dives deep into the process of implementing the SQUIRE model which leverages deep learning techniques to accurately translate user commands expressed in natural language into SQLite queries, enabling intuitive interaction with relational databases. Herein, we detail the methodological framework used in the process of implementation, algorithmic strategies and technical nuances that underpin the operationalization of this transformative system. Through a comprehensive examination of the implementation process, we aim to elucidate the practical considerations and engineering decisions critical and necessary to achieve the desired performance and robustness in real-world scenarios.

## 2 Used frameworks, languages and tools

### 2.1 Python v3.12

Python is an interpreted programming language at a high level that prioritizes readability and simplicity. Python was developed by Guido van Rossum and was first made available in 1991. It supports procedural, object-oriented, and functional programming paradigms. Its design philosophy makes extensive use of whitespace and has a simple syntax that makes the code easier to read, which encourages the production of clear, succinct code.

Python is versatile for a wide range of applications, from web development and data analysis to artificial intelligence and scientific computing, thanks to its large standard library and thriving third-party package ecosystem. Rapid development and prototyping are made easier by the language's dynamic typing and automatic memory management. [45]

### 2.2 TensorFlow v2.16

TensorFlow is an open-source machine learning framework developed by the Google Brain team and released in 2015. It provides a comprehensive, flexible ecosystem of tools, libraries, and community resources that enable researchers and developers to build and deploy machine learning models efficiently. TensorFlow is designed to support a range of tasks, from simple linear regression to complex neural networks, facilitating both research and production-grade applications.

TensorFlow functions fundamentally on data flow graphs, wherein edges and nodes, respectively represent multi-dimensional data arrays (tensors) that flow between mathematical processes. TensorFlow is extremely scalable and performant because to its graph-based architecture, which enables efficient computing on a variety of platforms, including CPUs, GPUs, and TPUs.

TensorFlow is widely used in both academic research and industry due to its versatility and strong community support. It enables researchers to prototype and experiment with new models, while also providing the robustness and scalability required for deploying machine learning applications in production environments. Its extensive documentation, tutorials, and active community forums further facilitate learning and collaboration among practitioners at all levels. [46]

### 2.3 C# v10.0

C# is a modern, object-oriented, and type-safe programming language designed by Microsoft for building a wide range of applications on the .NET framework. It supports the principles of object-oriented programming (OOP), such as encapsulation, inheritance, and polymorphism, enabling developers to create modular and reusable code. The language ensures type safety, preventing unauthorized memory access and enhancing security. C# is component-oriented, facilitating the development of software from prefabricated parts, which promotes code reuse and manageability. It offers excellent interoperability with other languages and systems, including unmanaged code, making it possible to leverage existing libraries and components. C# supports scalable and updatable applications, making it suitable for small to enterprise-level systems. Its unified type system ensures that all types, including primitive types, inherit from a single root object, enhancing consistency and interoperability within the language. [47]

### 2.4 WPF framework with .NET Core 6.0

Windows Presentation Foundation (WPF) is a graphical subsystem by Microsoft for rendering user interfaces in Windows-based applications. It is part of the .NET framework and provides a consistent programming model for building applications with rich user experiences. WPF uses Extensible Application Markup Language (XAML) for defining and linking various interface elements, enabling developers and designers to work collaboratively [48]. Also, key features of WPF includes:

- Declarative programming with XAML: WPF leverages XAML, an XML-based language, to define UI elements and their relationships in a clear and concise manner, separating design from logic. [48]
- Data Binding: WPF provides robust data binding capabilities, allowing the UI to reflect changes in the underlying data model automatically and support complex data visualization. [48]
- Styling and templates: WPF allows developers to alter the design and behavior of controls while maintaining a uniform look and feel throughout the program. [48]
- Interoperability: WPF can interoperate with existing Windows Forms and Win32 code, allowing developers to integrate new WPF components into existing applications. [48]
- Event-Driven Programming: WPF supports an event-driven programming model, allowing developers to respond to user actions and system events efficiently. [48]

### 2.5 Machine Learning Training Utilities (MLTU) package

The MLTU package is a comprehensive Python library designed to streamline the machine learning workflow by providing a suite of tools for various stages of the pipeline. It offers functions and classes for data preprocessing tasks such as data cleaning, normalization, and transformation, which are essential for preparing raw data for modeling. For feature engineering, MLTU provides tools to generate and select informative and predictive variables. It supports the training of various machine learning models, integrating seamlessly with popular frameworks like TensorFlow, PyTorch, and scikit-learn. [49]

Additionally, MLTU includes utilities for evaluating model performance through various metrics and visualizations, aiding in the interpretation of a model's effectiveness. It facilitates hyper-parameter optimization, allowing users to fine-tune their models using techniques like grid search and random search. The package also assists in deploying trained models into production environments by offering tools for model serialization, versioning, and serving. Furthermore, MLTU provides visualization tools to support exploratory data analysis and result presentation, making it easier to communicate findings and insights. This comprehensive toolkit enhances productivity and consistency for machine learning practitioners. [49]

### 2.6 Tokenizers library

The Tokenizers library from Hugging Face is a high-performance, open-source library designed for efficient and flexible tokenization of text data in natural language processing (NLP) tasks. Implemented in Rust, the library offers exceptional speed and low memory usage, making it capable of processing large text datasets much faster than many other tokenization tools. [40]

The library supports multiple tokenization algorithms, including Byte-Pair Encoding (BPE), WordPiece, SentencePiece, and Unigram, catering to a wide range of NLP models and applications. This versatility ensures that Tokenizers can be adapted to various model requirements and text preprocessing needs. Additionally, the library offers extensive customization options, allowing users to tailor tokenization pipelines by adding special tokens, handling different languages, and applying various preprocessing steps like normalization and truncation. [40]

Additionally, Tokenizers provide compatibility and optimal performance with a variety of NLP frameworks by offering fine-grained control over pre-tokenization (dividing text into words or sub-words) and post-processing (adding special tokens required by specific models). All things considered, Hugging Face's Tokenizers library is a vital tool for text data preparation in contemporary NLP applications since it blends excellent performance with user-friendliness and wide compatibility. [40]

# 3 Data gathering

Generally, as the number of parameters in a model increases, so does its capacity to learn complex patterns from data. However, this increased capacity often requires a proportional increase in the amount of training data to prevent overfitting. The exact ratio of data to parameters can vary depending on factors such as the complexity of the task, the diversity of the dataset, and the architecture of the model. Hence, a safe rule of thumb is to have at least several times more data points than parameters in the model [50]. Which can be quite intensive and computationally expensive, and consequently requires high performance hardware. Furthermore, in order to achieve the intended large size and diversity of data it is a good-practice to use data from multiple datasets.

Additionally, out of the whole data we will extract only SELECT queries due the mentioned heuristic rule of training. Moreover, since there is a huge lack of every type of query other than SELECT then using them in the training process would cause an appearance of noise since there is no efficient data to use.

## 3.1 Used datasets

### 3.1.1 GRETELAI synthetic dataset

High-quality synthetic Text-to-SQL samples are abundant in the GRETLAI synthetic_text_to_sql dataset, and containing only SQLite queries, it has been created and produced with Gretel Navigator and made available under Apache 2.0... [51] Here is an example of an object in the set:


```
{
 "id": 39325,
 "domain": "public health",
 "domain_description": "Community health statistics, infectious disease tracking data, healthcare access metrics, and public health policy analysis.",
 "sql_complexity": "aggregation",
 "sql_complexity_description": "aggregation functions (COUNT, SUM, AVG, MIN, MAX, etc.), and HAVING clause",
 "sql_task_type": "analytics and reporting",
 "sql_task_type_description": "generating reports, dashboards, and analytical insights",
```

```
 "sql_prompt": "What is the total number of hospital beds in each state?",

 "sql_context": "CREATE TABLE Beds (State VARCHAR (50), Beds INT); INSERT INTO Beds (State, Beds) VALUES ('California', 100000), ('Texas', 85000), ('New York', 70000);",

 "sql": "SELECT State, SUM(Beds) FROM Beds GROUP BY State;",

 "sql_explanation": "This query calculates the total number of hospital beds in each state in the Beds table. It does this by using the SUM function on the Beds column and grouping the results by the State column."

}
```


Moreover, as mentioned prior to this, SQL has different types of queries in order to be able to perform a variety of operations on SQL databases. Hence, this dataset contains the following types of queries:

1. SELECT: approximately 90,000 queries
2. INSERT: approximately 3200 queries
3. DELETE: 3356 queries
4. DROP: approximately 500 queries
5. UPDATE: 2978 queries
6. ALTER: 30
7. CREATE: approximately 300 queries

We are concerned mostly in only three attributes out of the rest, which are the “sql_prompt”, “sql_context”, and the “sql” attribute. The sql_prompt attributes represents the natural language command, in addition to sql_context, which is the database’s schema. Lastly, the sql attribute represents the SQLite query corresponding to the given object.

The dataset includes 105,851 well distributed across domains samples in total with approximately 23 million tokens where 12 million of them are SQL tokens covering over 100 distinct domains. [51]

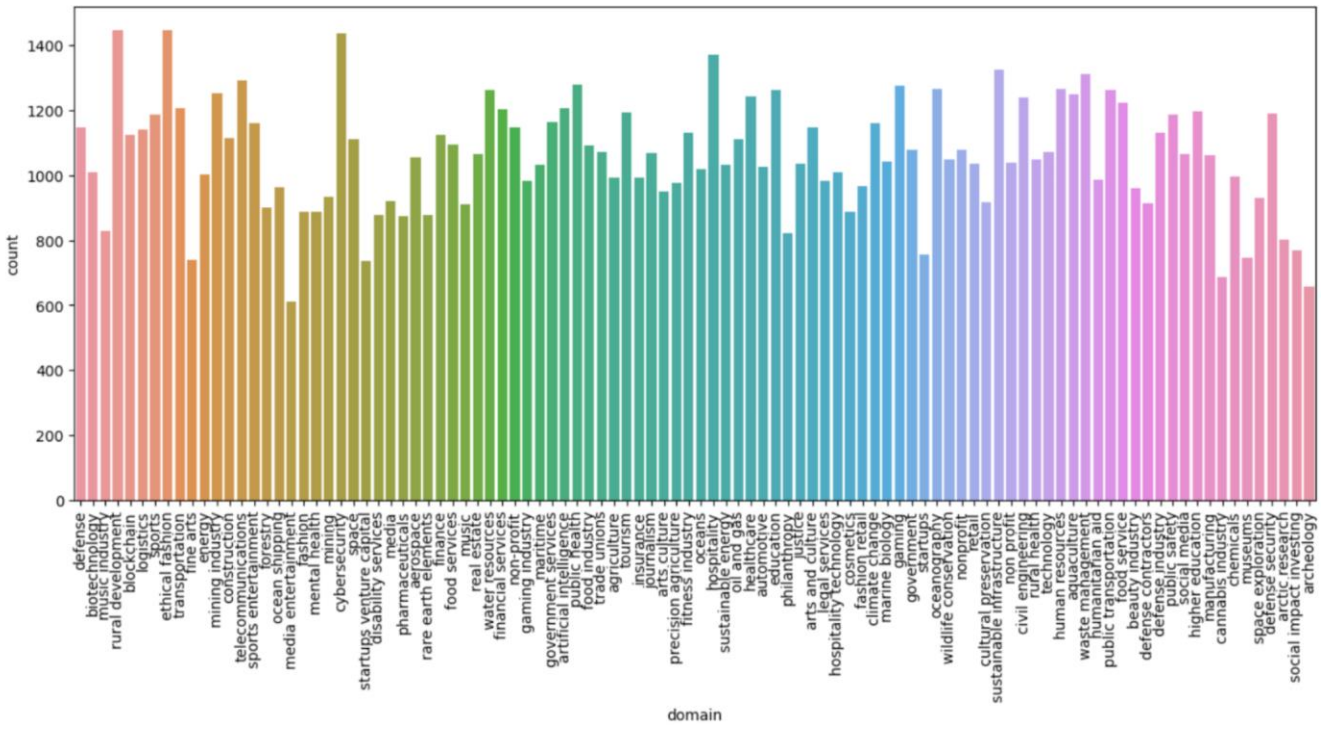


*Figure 15 Synthetic dataset domain distribution*

### 3.1.2 Clinton

Clinton/text-to-sql-v1 is a moderate size dataset containing 262,208 samples, consisting only of SELECT SQLite queries [52], where each single sample is formulated like the following object:

```
{
   "instruction": “Name the home team for carlton away team”,
    "input": "CREATE TABLE table_name_77 ( home_team VARCHAR, away_team VARCHAR )",
    "response": "SELECT home_team FROM table_name_77 WHERE away_team = ‘carlton’",
    "source": "sql_create_context",
   "text": "### Instruction: Name the home team for carlton away team ### Input: CREATE TABLE table_name_77 ( home_team VARCHAR, away_team VARCHAR ) ### Response: SELECT home_team FROM table_name_77 WHERE away_team = ‘carlton’"

}
```

In our particular project, we only focus on three attributes out of the mentioned above, which namely are “instruction” being the attribute representing the natural language command, “input” which is the database schema, and “response” which represents the corresponding SQL query. [52]

### 3.2 Final synthetic dataset and data processing

The provided data in the previously mentioned datasets are not well suited and formulated to use in our specific task, which requires a phase of data cleaning, augmentation, and preprocessing. Starting with the attributes themselves, since we only need a portion of each dataset’s attributes, which represents mainly the natural language command, database schema, and the corresponding SQLite query, then it necessary to extract the needed marked ones from each object to make a new suitable set of samples. After that the three main intended attributes were extracted in a new set of samples, now they need to follow the model’s specified convention which is formulated like:

- Input: database schema + natural language command
- Output: SQLite query

Furthermore, the first two resulting attributes needs to be concatenated in every object with a specific manner in order to become the input sequence for training, whilst the output sequence, namely the SQLite query is already in place. Moreover, we get a dataset containing approximately 360,000 samples formulated like the following:

```
{
    [
        "input": “CREATE TABLE table_name_77 ( home_team VARCHAR, away_team VARCHAR )—prompt: Name the home team for carlton away team”,
        "output": " SELECT home_team FROM table_name_77 WHERE away_team = ‘carlton’"
    ],
}
```

Sequences lengths vary in a range from 80 to 5012 in the case of input sequences, and output sequences vary in the range of 30 to 3500, noting that the length calculated is the number of characters in a sequence.

Now at this point the data is collected although not yet ready to be exploited, due to it not being clean in several ways. A main one of which is the existence of non-English characters whom can cause a major deflect as the tokenizers do not recognize them, and in addition to that, they would cause noise in the data since the absolute major part of it is mainly English only, hence that such objects needs to be removed from the set. Finally, after collecting all the data needed and cleaning/preprocessing it, we end up with a dataset whose samples are formulated like the object above, consisting of approximately 300,000 samples and ready to be used on training.

# 4 Training

Training a transformer model can be a very intricate and intensive task to accomplish, and due to this fact, a set of tools and variables including the batch size, the number of epochs, and the warm-up algorithm, needs to be studied thoroughly and experimented for a sufficient amount of times.

## 4.1 Warm up

Model warmup is a technique used in machine learning and deep learning to gradually increase the learning rate at the beginning of the training process. This helps to stabilize training by allowing the model to start learning with smaller updates, which can prevent large, potentially destabilizing weight updates that might occur if the learning rate were set high initially. Warmup typically involves a period where the learning rate starts at a very low value and gradually increases to a predefined maximum value, after which the learning rate scheduling can follow the standard policy (e.g., step decay, cosine annealing, etc.). [53]

The used schedule in this particular case is CosineDecay, which reduces the learning rate according to the cosine of the training progress, starting from an initial learning rate and gradually decreasing to a minimum learning rate. This schedule can help in maintaining longer training with relatively high learning rates and allows smoother transitions to lower learning rates. [53]

For an initial learning rate $\eta_0$, and a minimum learning rate $\eta_{\min}$, the learning rate $\eta(t)$ at a given time step $t$ (out of a total of $T$ steps) can be computed using the cosine decay formula:

$$\eta(t) = \eta_{min} + \frac{1}{2}(\eta_0 - \eta_{min})(1 + \cos(\frac{t\pi}{T}))$$ [53]

Where:

- $\eta_0$: Initial learning rate.

- $\eta_{min}$: Minimum learning rate.
- $t$: Current time step.
- $T$: Total number of time steps for the training process.
- $\cos(\frac{t\pi}{T})$: The cosine function that modulates the learning rate.

Finally, the used CosineDecay has been instantiated using the WarmupCosineDecay class from the MLTU [49] package, with the following schedule parameters:

- Initial learning rate: 0.00001.
- Final learning rate: 0.0001.
- Learning rate after warmup: 0.0005.
- Warmup epochs: 4.
- Decay epochs: 18.

### 4.2 Data batching and schedule

Data batching same as the schedule are significantly influenced with the particular used hardware since the data batching depends on how much the memory can handle, and scheduling the epochs also depends on the power of processing, meaning that these values can be quite relative and needs to be experimented with a variety of values to determine the optimal set. Consequently, after a sequence of experiments the set of values are determined like the following:

- Batch size: 16.
- Number of training epochs: 30.

Although, it is important to note that the EarlyStopping, and AutoSave callback have been implemented, hence the training doesn't have to go through all of the epochs but can actually stop prior to the last possible epoch.

### 4.3 Hardware

Performative hardware is an essential asset in the process of training AI models especially transformers as they are so data-hungry hence can be very intensive to train. This particular aspect prompts the severe need for powerful machines and exploit their abilities in order to make better use of the available time and resources. Moreover, the used hardware in this project is a very humble consisting of a 7$^{th}$ generation i7 processor, 16GB of ram, and a GTX1060 NVIDIA GPU, all put in an MSI laptop, model GP62. It is also necessary to note that GPUs are designed with a large number of cores optimized for parallel processing, making them ideal for operations involving large matrices and tensors commonly found in deep learning. Mathematically, many deep learning operations involve matrix multiplications, convolutions, and element-wise operations. [55]

In theory, The time complexity for matrix multiplication for example on a single-threaded CPU is O($n^3$) for cubic matrices, while GPUs can reduce the effective time complexity through massive parallelism, approaching O($n^2$) in practice for large matrices due to simultaneous computations [55]. Consequently, the training has been executed on the GPU using CUDA 11.5 [54], thanks to TensorFlow's support for GPU computing.

### 4.4 Optimizer

Adaptive Moment Estimation with Weight Decay (AdamW) is an optimization algorithm designed to update network weights iteratively based on training data. It builds on the Adam [57] optimizer by incorporating a decoupled weight decay term, which helps improve generalization by regularizing the model parameters. This modification addresses some of the shortcomings of the original Adam [57] optimizer in handling weight decay, leading to better performance in many scenarios. [56]

The update rules for AdamW are:

1. Compute biased first moment estimate:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \text{ [56]}$$

2. Compute biased second raw moment estimate:

$$v_t = \beta_2 v_{v-1} + (1 - \beta_2) g_t^2 \text{ [56]}$$

3. Compute bias-corrected first moment estimate:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t} \text{ [56]}$$

4. Compute bias-corrected second raw moment estimate:

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t} \text{ [56]}$$

5. Update parameters:

$$\theta_{t+1} = \theta_t - \eta\left(\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \varepsilon} + \lambda\theta_t\right) \text{ [56]}$$

Where:

- $g_t$ is the gradient at time step $t_t$.
- $\beta_1$ and $\beta_2$ are decay rates for the moment estimates.
- $\eta$ is the learning rate.
- $\epsilon$ is a small constant to prevent division by zero.
- $\lambda$ is the weight decay coefficient.

# 5 Evaluation metrics

## 5.1 Performance metrics

To evaluate the effectiveness of the model, several key performance metrics has been employed. These metrics are crucial in understanding the model's learning capacity, accuracy, and generalization to unseen data. By assessing these metrics, we can gauge the model's performance comprehensively and identify areas for potential improvement. The following section provides detailed definitions and insights into each of these metrics: masked loss, masked accuracy, masked validation loss, and masked validation accuracy.

Masked sections are certain tokens within a sequence that are intentionally masked during the model training or evaluation. These tokens are temporarily replaced with a special masking token to prevent the model from directly witnessing them, forcing it to predict their identities based on contextual information. This masking method simulates instances in which certain information is missing or concealed, enabling the model to acquire resilient representations and dependencies from the input data. [58]

1. Masked accuracy: Masked accuracy is a performance metric indicating the proportion of correctly predicted masked tokens during training. It assesses the model's ability to accurately infer the masked portions of the input sequence based on contextual information [58]. Mathematically, it is expressed as:

$$\text{Masked accuracy} = \frac{Number\ of\ correctly\ predicted\ masked\ tokens}{Total\ number\ of\ masked\ tokens} \times 100\% \text{ [58]}$$

2. Masked loss: Masked loss, denoted as $L_{masked}$, is a key training metric in transformer models, computed as the average cross-entropy loss over all masked tokens. It quantifies the disparity between the model's predicted probability distribution and the true distribution of the masked tokens in the input sequence [24]. Mathematically, it can be represented as:

$$L_{masked} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{T} 1(m_{i,j})\ \log(p_{i,j}) \text{ [24]}$$

3. Masked validation accuracy: Masked validation accuracy quantifies the proportion of correctly predicted masked tokens in the validation dataset. It provides insights into the model's ability to generalize to new, unseen instances by accurately predicting masked tokens. The masked validation accuracy is computed similarly to the training masked accuracy but applied to the validation dataset. [58]
4. Masked validation loss: Masked validation loss represents the average discrepancy between the model's predicted probability distribution and the true distribution of masked tokens in the validation dataset. It serves as a measure of the model's generalization performance to unseen data. The masked validation loss is computed similarly to the training masked loss but applied to the validation dataset. [24]

## 6 Results

After that the training finished, a set of results can be observed which encapsulates the outcomes of various experiments conducted to fine-tune and optimize the model's parameters. These findings provide insights into the model's accuracy, efficiency, and overall effectiveness in addressing the targeted tasks:

1. Number of executed epochs: 20.
2. Training masked accuracy: 0.90.

3. Training masked loss: 0.49.
4. Validation masked accuracy: 0.92.
5. Validation masked loss: 0.44.

Overall, the transformer model underwent an extensive training process over 20 epochs, resulting in impressive performance metrics. The training masked accuracy reached 0.90 (90%), indicating that the model effectively learned the relationships and context within the training data. This is further supported by a relatively low training loss of 0.49, suggesting a good fit to the training data. Notably, the model demonstrated strong generalization capabilities, achieving a slightly higher validation masked accuracy of 0.92 (92%) and a lower validation masked loss of 0.44.

These results highlight the model's robustness and its ability to perform well on unseen data, underscoring its potential for practical application in relevant tasks.

### 6.1 Host user-friendly application

To facilitate user interaction with the transformer model, we developed a C# WPF application. This application provides a user-friendly interface for leveraging the model's capabilities, allowing users to input data, receive predictions, and visualize results seamlessly.

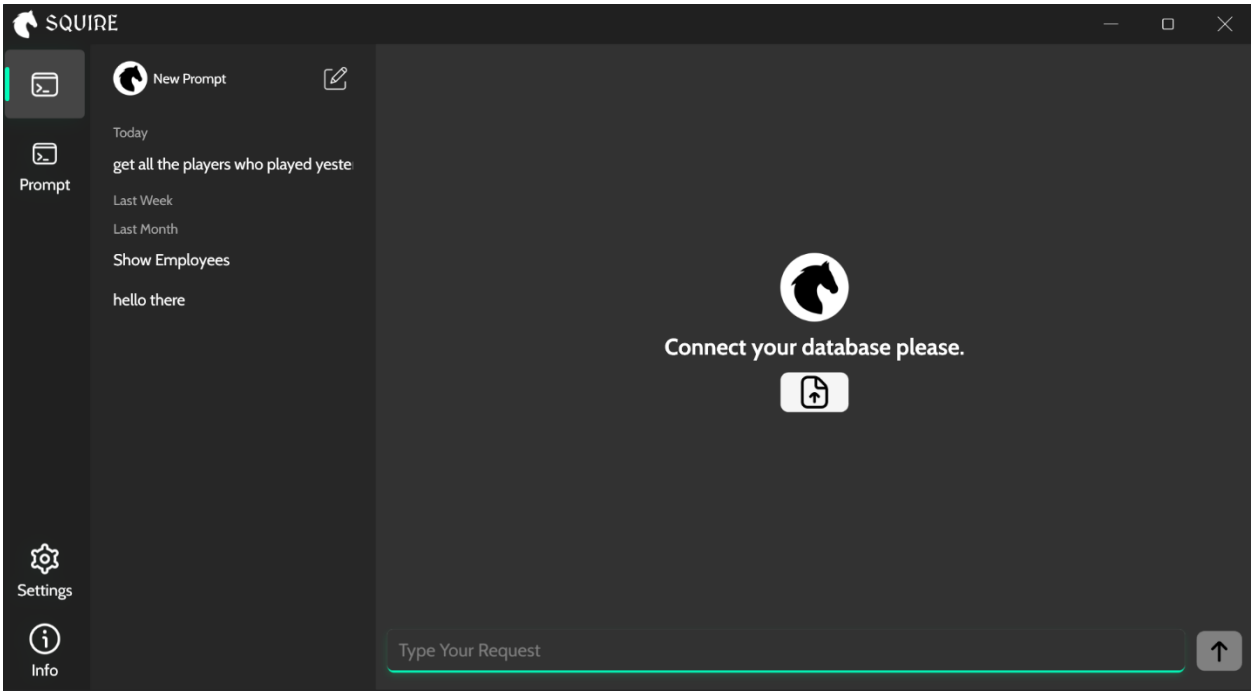


*Figure 16 Main application interface*

Now that the application is open, all that is left to do is to choose the intended SQLite database to operate on, all with seamless ease by clicking the file upload button:

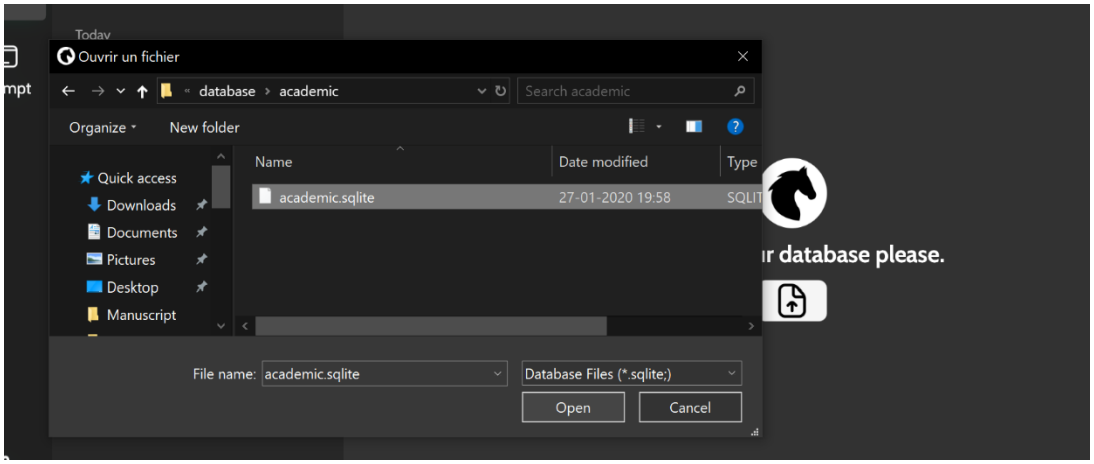


*Figure 17 Example of connecting a database to the application*

Since some SQL databases can be protected with a password, it has to be provided in order to gain access:

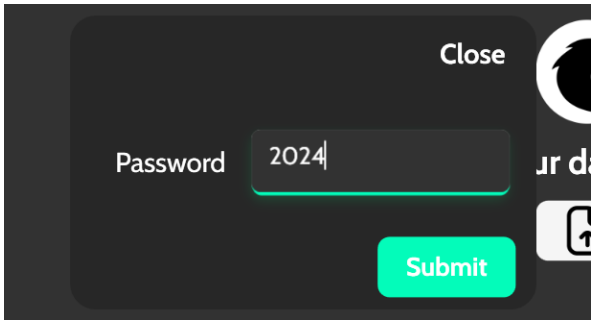


*Figure 18 entering the password for the database*

## 6.2 Example conversions

After that the database is connected and accessed successfully, it is simply ready to be operated on. Like chatbots, the natural language command should be written in the text box bellow and sent, after that the model gets the command it concatenates it with the corresponding database schema and then start the inference and finally reply with the resulting query:

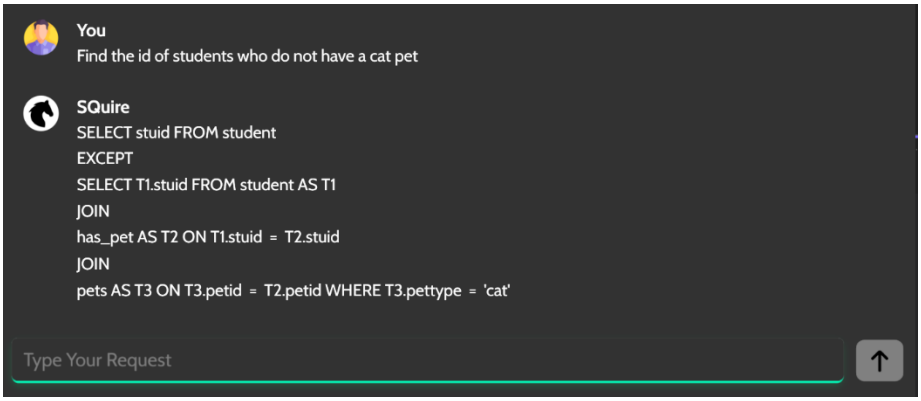


*Figure 19 first conversion example*

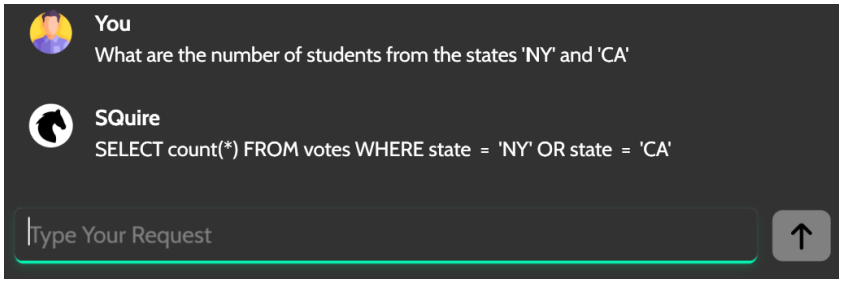


*Figure 20 second conversion example*

## 7. Conclusion

In this chapter, we have detailed the implementation process of our model. Beginning by presenting the various tools used during the coding phase, as well as the datasets leveraged, emphasizing their features and relevance to the project, then combining them into one moderate size synthetic dataset. We show-cased the results, including user interfaces and example conversions of natural language commands into SQL queries. In addition, to outlining the evaluation metrics employed to assess the system's performance.

Despite the good results the model ended up having, it is not well sufficient to the point to call it an NLDBI due to the fact that it can handle only SELECT SQL queries because of the lack of training data modeling other kinds of queries. Hence, the one aspect with most of the priority to be included in future updates is the inclusion of the rest of SQLite queries in order to enable the model to handle any kind of database management command. Moreover, much better results and generalization could have been achieved if the training data were at least 7 times as much as the parameters amount according to the heuristics of training transformers. In another words, transformer models can be very efficient in NLP tasks in particular but they are very data-hungry and can require very powerful hardware in order to achieve acceptable results which can pose hard challenges most of the times.

# GENERAL CONCLUSION

In this project, we proposed a model named SQUIRE. The aim of this model was to provide an intuitive and user-friendly means of interaction, serving as an NLDBI (Natural Language Database Interface) for relational databases, allowing users to query databases without requiring a complete understanding of SQL.

To achieve this, we began by studying and reviewing the extensive field of artificial intelligence known as natural language processing (NLP), along with its key applications, techniques, and approaches. This was essential to gain a comprehensive understanding of this specific area of science and to clarify the techniques and framework that would later be used in our work.

Subsequently, we presented a conceptual model to establish this model, while elucidating the steps covered by this model, including the algorithms and pipelines used, as well as the training and testing data. This final task could not be accomplished without the use of state-of-the-art measures to evaluate our work, including usability and performance metrics mentioned in this latter part.

We conclude that the results obtained are very promising given the difficulty and complexity of the domain of application. For future improvements, we suggest the incorporation of the remaining SQLite queries to enable the model to handle any type of database management command. Besides, we suggest, implementing this model as an API capable of working with different information systems and querying their databases to allow non-experts to leverage their data without the need to learn formal SQL language. Additionally, this would enable the execution of non-predefined queries to information systems, expanding the range of knowledge exploitation included in the databases.

## Bibliography

[1] https://www.ibm.com/topics/natural-language-processing. «What is natural language processing (NLP)? ». 2024/01/30.

[2] Gobinda G. Chowdhury, «Introduction to Modern Information Retrieval», Third Edition 2010, page 1.

[3] Hobbs, Jerry R., and Ellen Riloff. «Handbook of Natural Language Processing Information Extraction», Chapter 21, 2010, pages 1-2.

[4] Dimitrios Rafailidis, Yannis Manolopoulos. «The Technological Gap Between Virtual Assistants and Recommendation Systems » 2019, pages 1-2.

[5] https://en.wikipedia.org/wiki/Machine_translation. «Machine Translation». 2024/03/16.

[6] Walaa Medhat, Ahmed Hassan, Hoda Korashy. «Sentiment analysis algorithms and applications: A survey» 2013, pages 1-2.

[7] https://en.wikipedia.org/wiki/Automatic_summarization. «Automatic Summarization». 2024/03/16.

[8] Manning, C. D., Raghavan, P., & Schütze, H. «Introduction to Information Retrieval» 2008, Cambridge University Press.

[9] Iyer, Sheth, and Larson. «Natural Language Interfaces to Databases – An Introduction». 1993.

[10] Jackson, P., & Moulinier, I. «Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization. John Benjamins Publishing Company». 2007.

[11] Mrs. Neelu Nihalani, Dr. Sanjay Silakari, Dr. Mahesh Motwani. «Natural language Interface for Database: A Brief review» Vol. 8, Issue 2, 2011 Mar, pages 600-601.

[12] Janet O. Olaleke, Olaronke G. Iroju. «A Systematic Review of Natural Language Processing in Healthcare». 2015, page 44.

[13] Grefenstette, G. «Syntactic Wordclass Tagging. Text, Speech and Language Technology». Vol9, 1999, page 117.

[14] Alebachew Chiche, Betselot Yitagesu. «Part of speech tagging: a systematic review of deep learning and machine learning approaches». 2022, page 2.

[15] Vimala Balakrishnan, and Ethel Lloyd-Yemoh. «Stemming and lemmatization: A comparison of retrieval performances». 2014, pages 174-175.

[16] Andrei Mikheev, Marc Moens, Claire Grover. «Named Entity Recognition without Gazetteers». 1999, HCRC Language Technology Group, University of Edinburgh, page 1.

[17] Sandra Kübler, Ryan McDonald, Joakim Nivre. «Dependency Parsing», University of Toronto, 2009.

[18] Daniel Jurafsky, James H. Martin. «Speech and Language Processing» chapter 17 Context-Free Grammars and Constituency Parsing, 2000.

[19] Dastan Hussen Maulud, SubhiR. M.Zeebaree, Karwan Jacksi, Karzan Hussein Sharif, Mohammed A.M.Sadeeq. «A State of Art for Semantic Analysis of Natural Language Processing». Qubahan Academic Journal, 2021.

[20] Zexuan Zhong, Danqi Chen. «A Frustratingly Easy Approach for Entity and Relation Extraction». Department of Computer Science Princeton University, 2021.

[21] Yiren Wang. «Representation Learning: Algorithms and Models». Lecture 13: Language Modeling, University of Illinois, 2017.

[22] https://en.wikipedia.org/wiki/Word_embedding. «Word embedding». 2024/04/15.

[23] Andrea Galassi , Marco Lippi , and Paolo Torroni. «Attention in Natural Language Processing». IEEE transactions on neural networks and learning systems, VOL. 32, NO. 10, OCTOBER 2021.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. «Attention is all you need». 2017.

[25] Juan Ramos. «Using TF-IDF to Determine Word Relevance in Document Queries». Department of Computer Science, Rutgers University, 2003.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. «Deep Learning», Cambridge Massachusetts, 2017.

[27] Samsul Arifin , Andyan Kalmer Wijaya , Rinda Nariswari , I Gusti Agung Anom Yudistira , Suwarno , Faisal , Diah Wihardini. «Long Short-Term Memory (LSTM): Trends and Future Research Potential», International Journal of Emerging Technology and Advanced Engineering, 2023.

[28] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. «Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling», NIPS, 2014.

[29] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. «The Elements of Statistical Learning», 2009, pages 9-29.

[30] Christopher M. Bishop. «Pattern Recognition and Machine Learning», 2006.

[31] https://en.wikipedia.org/wiki/Natural_language_processing. «Natural Language Processing», 2024/01/30.

[32] Setsuo Arikawa, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, Shuichi Fukamachi, Takuya Kida, Yusuke Shibata. «Byte Pair Encoding: a text compression scheme that accelerates pattern matching», 1994.

[33] John Richardson, Taku Kudo. «SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing», 2018.

[34] https://spacy.io/models/en. «English models», 2024/05/23.

[35] Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, Denny Zhou. «Fast WordPiece Tokenization», Google Research, Mountain View, CA, 2021.

[36] https://en.wikipedia.org/wiki/Seq2seq. «Seq2seq», 2024/05/23.

[37] Jim Melton, Alan R. Simon. «SQL: 1999: Understanding Relational Language Components», Elsevier Science, 2002.

[38] Emiel Hoogeboom1, Didrik Nielsen2, Priyank Jaini1, Patrick Forré, Max Welling. «Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions», University of Amsterdam, UvA Bosch Delta lab, 2021.

[39] Https://huggingface.co/docs/tokenizers/en/index. «Tokenizers», 2024/05/29.

[40] https://huggingface.co/docs/transformers/en/main_classes/tokenizer. «Tokenizer», 2024/05/29.

[41] https://huggingface.co/docs/tokenizers/en/api/pre-tokenizers. «Pre-Tokenizers», 2024/05/29.

[42] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. «Sequence to Sequence Learning with Neural Networks», Google, 2014.

[43] https://unicode.org/reports/tr15/. «UNICODE NORMALIZATION FORMS», 2024/05/31.

[44] huggingface - StripAccents normalizer.. «Normalizers», 2024/05/31.

[45] https://www.python.org. «Python», 2024/06/01.

[46] https://www.tensorflow.org. «TensorFlow», 2024/06/01.

[47] https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview. «A tour of the C# language», 2024/06/01.

[48] https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0. «Desktop Guide WPF .NET», 2024/06/01.

[49] https://pypi.org/project/mltu/. «MLTU - Machine Learning Training Utilities», 2024/06/01.

[50] YI TAY, MOSTAFA DEHGHANI, DARA BAHRI, DONALD METZLER. «Efficient Transformers: A Survey», Google Research, 2019.

[51] Meyer, Yev and Emadi, Marjan and Nathawani, Dhruv and Ramaswamy, Lipika and Boyd, Kendrick and Van Segbroeck, Maarten and Grossman, Matthew and Mlocek, Piotr and Newberry, Drew. «Synthetic-Text-To-SQL», 2024, gretelai/synthetic_text_to_sql · Datasets at Hugging Face.

[52] Clinton/Text-to-sql-v1 · Datasets at Hugging Face. «Clinton text-to-sql-v1», 2024/06/01.

[53] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, Richard Socher. «A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation», 2018.

[54] https://developer.nvidia.com/cuda-toolkit. «CUDA Toolkit», 2024/06/03.

[55] ERIC LIND, ÄVELIN PANTIGOSO. «A PERFORMANCE COMPARISON BETWEEN CPU AND GPU IN TENSORFLOW», EXAMENSARBETE INOM TEKNIK, GRUNDNIVÅ, 15 HP STOCKHOLM, 2019.

[56] ILYA LOSHCHILOV, FRANK HUTTER. «DECOUPLED WEIGHT DECAY REGULARIZATION», 2017.

[57] DIEDERIK P. KINGMA, JIMMY LEI BA. «ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION».

[58] JACOB DEVLIN, MING-WEI CHANG, KENTON LEE, KRISTINA TOUTANOVA. «BERT: PRE-TRAINING OF DEEP BIDIRECTIONAL TRANSFORMERS FOR LANGUAGE UNDERSTANDING », 2018.

**ملخص:**

في ألعصر الرقمي الحالي، تواجه المؤسسات تحدي إدارة مجموعات بيانات ضخمة، مما يستلزم وجود أدوات متطورة لتحليل البيانات. بينما يعتبر لغة الاستعلام المهيكلة (SQL) خيارًا قويًا للتفاعل مع قواعد البيانات، إلا أن تعقيدها يشكل عقبات، خاصة بالنسبة للمستخدمين غير التقنيين. يقدم هذا المشروع نموذجSQUIRE ، الذي يهدف إلى تقديم واجهة بيانات طبيعية مفهومة (NLDBI) لقواعد البيانات العلاقيةSQL ، بهدف تمكين المستخدمين من إنشاء استعلامات SQLمتنوعة بلغة طبيعية وبدون الحاجة إلى معرفة وافية بـSQL ، مما يعزز المرونة والسهولة في استرداد البيانات .

**الكلمات المفتاحية:** SQL، قواعد المعطيات، SQUIRE، NLDBI، Python

---

**Abstract:**

In today's digital era, organizations grapple with the challenge of managing vast datasets, necessitating sophisticated tools for data analysis. While Structured Query Language (SQL) serves as a robust choice for database interaction, its complexity poses barriers, especially for non-technical users. This project introduces SQUIRE, a model designed to bridge this gap by offering an intuitive Natural Language Database Interface (NLDBI) for SQL relational databases. SQUIRE aims to democratize data access, empowering users to generate diverse SQL queries without extensive SQL knowledge, thereby enhancing flexibility in data retrieval.

**Key words:** SQL, Data Base, SQUIRE, NLDBI, Python

---

**Résumé :**

À l'ère numérique d'aujourd'hui, les organisations font face au défi de gérer de vastes ensembles de données, nécessitant des outils sophistiqués pour l'analyse des données. Alors que le langage de requête structuré (SQL) est un choix robuste pour l'interaction avec les bases de données, sa complexité pose des obstacles, notamment pour les utilisateurs non techniques. Ce projet présente SQUIRE, un modèle conçu pour combler cette lacune en offrant une interface intuitive de base de données en langage naturel (NLDBI) pour les bases de données relationnelles SQL. SQUIRE vise à démocratiser l'accès aux données, permettant aux utilisateurs de générer diverses requêtes SQL sans une connaissance approfondie du SQL, améliorant ainsi la flexibilité dans la récupération des données.

**Mots clés :** SQL, Base de données, SQUIRE, NLDBI, Python