Using Domain Specific Language and Sequence to Sequence models as a hybrid framework for a Natural Language Interface to a Database solution

A Thesis Submitted for the Degree of Doctor of Philosophy

by

Richard Skeggs

Department of Computer Science, Brunel University London

March 2023

Declaration

I, Richard Skeggs, hereby declare that this thesis and work presented in it is entirely my own. Some of the work has been previously published in journal or conference papers and this has been mentioned in the thesis. Where I have consulted the work of others this is always clearly stated.

Acknowledgement

There is an old proverb which states that a journey of a thousand miles begins with a single step. The completion of this thesis has been a long-held ambition of mine for a number of years. I had an idea born out of a common problem that I had continually encountered while working in data projects. I had approached Brunel University with the idea and after some manipulation and tweaking of the original idea I eventually had a plan that would actually make a worthwhile project. For all the help, guidance and support that I have received from Dr. Stasha Lauria I will be eternally grateful. Without his support and guidance this journey would have been far harder and may not have even been possible.

Having begun the journey with my idea of what the project should deliver and no support from an employer the only way I was ever going to reach my goal would have been with the support of my family. Having just bought a house that requires a lot of work, the demand to work on the house and work on this research project has been a major conflict. Understanding from the family has been paramount. The hours I have spent on the keyboard over the course of the evenings during the week and the days during the weekend has required a special type of patience from the family. With that in mind I would like to thank my mother, my partner and all our children. Your support has been essential allowing me the time and space required to complete this project.

On this journey there has also been the added complexity of trying to complete this project within the scope of a global pandemic which has caused a massive upheaval to society, healthcare, work and study. Apart from the usual issues my partner has been on the front line of covid which has put extra pressure on me to support our children. The support from Brunel University Computer Science Department has been invaluable in allowing me the time and space to get this project over the line. In particular I would like to thank Professor Zidong Wang, Dr Stephen Swift and Professor Xiaohui Liu for their assistance and the input that they provided proved to have been invaluable.

Abstract

The aim of this project is to provide a new approach to solving the problem of converting natural language into a language capable of querying a database or data repository. This problem has been around for a while, in the 1970's the US Navy developed a solution called LADDER and since then there have been an array of solutions, approaches and tweaks that have kept the research community busy. The introduction of electronic assistants into the smart phone in 2010 has given new impetus to this problem.

With the increasingly pervasive nature of data and its ever expanding use to answer questions within business science, medicine extracting data is becoming more important. The idea behind this project is to make data more democratised by allowing access to it without the need for specialist languages. The performance and reliability of converting natural language into structured query language can be problematic in handling nuances that are prevalent in natural language. Relational databases are not designed to understand language nuance.

This project introduces the following components as part of a holistic approach to improving the conversion of a natural language statement into a language capable of querying a data repository.

- The idea proposed in this project combines the use of sequence to sequence models in conjunction with the natural language part of speech technologies and domain specific languages to convert natural language queries into SQL. The approach being proposed by this chapter is to use natural language processing to perform an initial shallow pass of the incoming query and then use Google's Tensor Flow to refine the query with the use of a sequence to sequence model.

- This thesis is also proposing to use a Domain Specific Language (DSL) as part of the conversion process. The use of the DSL has the potential to allow the natural language query to be translated into more than just an SQL statement, but any query language such as NoSQL or XQuery.

Table of Contents

List of Figures

List of Code Samples

List of Tables

1. Introduction

The aim of this project is to provide an original approach to solving the problem of converting natural language into a language capable of querying a database or data repository. This problem has been around for a while, in the 1970's the US Navy developed a solution called LADDER and since then there have been an array of solutions, approaches and tweaks that have kept the research community busy. The introduction of electronic assistants into the smart phone in 2010 has given new impetus to this problem.

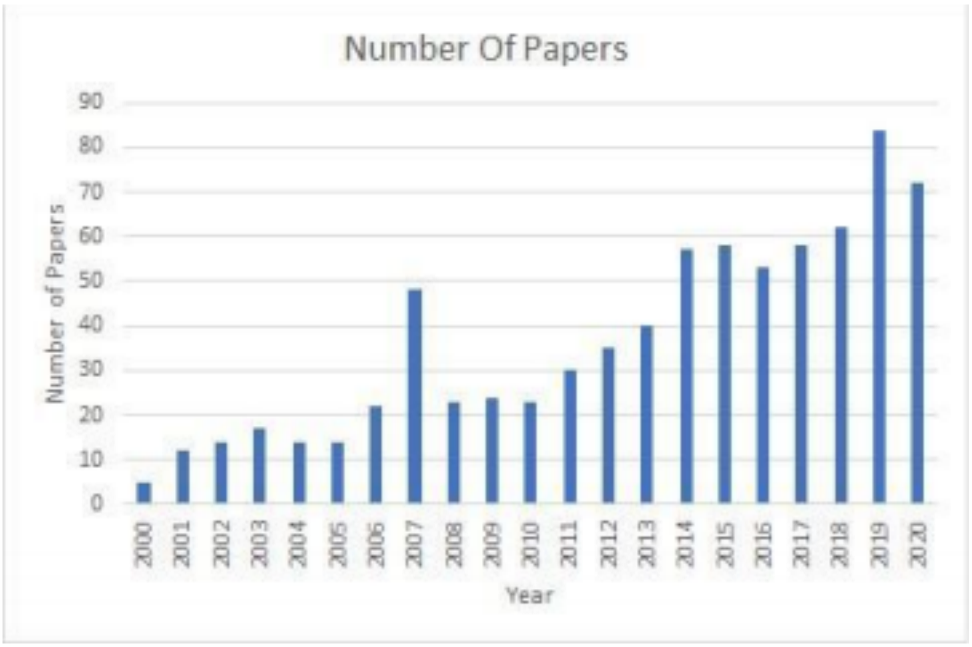


Figure 1-.1: Shows the number of papers on the subject NLIDB between 2000 and 2019 that were searchable via google scholar.

As can be seen from Figure 1 even taking a narrow search term such as *NLIDB* it is possible to see that the trend in research for this topic has been upwards. It must also be remembered that research in this field is not just concentrated on the simple term NLIDB, but has also included areas such as semantic parsing, sequence to sequence models and augmented memory. The increased depth in this field potentially shown by the increased number of published papers and the increased breadth highlighted by the number of related research topics would indicate the growing importance of this topic.

This project looks at simplifying the process of converting a natural language statement into a language capable of querying a database. The project also looks at creating a simplified interface into accessing algorithms that can convert natural language into a structured query language. To achieve both these goals this project introduces two novel approaches.

1. The use of part of speech or shallow parsing to extract keywords from the incoming natural language statement. The identified keywords are then compared to objects within the underlying database to facilitate the conversion from natural language to a language capable of querying a database.

2. The creation of a domain specific language (DSL) to help create a complete solution to the problem of converting natural language into a language capable of querying a database. This project proposes using an external DSL based on the Bloom knowledge taxonomy to help facilitate the conversion from natural language to SQL.

This project looks at using both flavours of DSL. Initially this chapter highlights the possibility of creating an external DSL that is capable of translating a natural language statement to a structured query language (SQL) statement for querying a data repository. The idea is that the DSL can be used as a front end interface allowing parsing techniques and translation models from other research projects to be incorporated into the DSL. None of the projects that were reviewed provided an interface to using a NLIDB solution. To that extent this project introduces a novel approach to converting natural language to SQL, there are more details on this subject in the following chapters. This new parsing approach is then incorporated into the DSL as a demonstration of how to ensure the new language can be extended to incorporate new and expanding techniques. The project also introduces the concept of an internal DSL that can be used as part of an approach to convert natural language to a structured query language. The implementation of the internal DSL is not fully realised within the project as it is beyond the scope of this project. The implementation of the internal DSL is something that can be investigated as part of a further project as it is not core to the successful competition of this project.

This project starts with an introduction to the fields of natural language interface to a database and domain specific languages. This includes the current state of research and development in both fields as well as an introduction of how these fields are being applied within this project. This is then followed by the proposed new multi-phased approach of using shallow parsing, sequence to sequence models to extract the salient words from the incoming natural language statement to produce a language capable of querying a database. The final section in this project then looks at the development and use of the domain specific language to provide a simplified interface into using the algorithm proposed by this project.

1.1. Motivation

The growth in data has been the business story for the last eleven years. The table below shows the growth in data marked in both the amount of data being generated in zettabytes as well as the revenue value of data in billions of United States Dollars to global business. This growth in the production and usage of data along with the expected increase in revenues for organisations has fuelled a demand for tools to extract data allowing organisation to make value judgements based on data.

This research project was motivated by the need to democratise the search capabilities of large data repositories. Currently to perform a search on a dataset, specialist skills are required. The Structured Query Language (SQL) is a powerful tool that can be used to extract data. The problem is that it takes a high degree of technical competence to use SQL properly. There are tools that can take a natural language statement and convert it into a language capable of querying a database. However as will be discussed in Chapter 2 these tools provide an interface into a repository but their performance in terms of speed of conversion and accuracy in extracting the desired search results from the underlying repository is not at the required level of performance. The aims of this project are to provide a conversion algorithm that can take the natural language input and provide returned results that closely match the expected results of the user in a timeframe that is acceptable to the user. With previous experience in providing data repositories most users require results in under five seconds. Along with the improve improvements in time and accuracy this project will also provide a common extensible interface into the algorithm as well as allowing other algorithms to use the same interface.

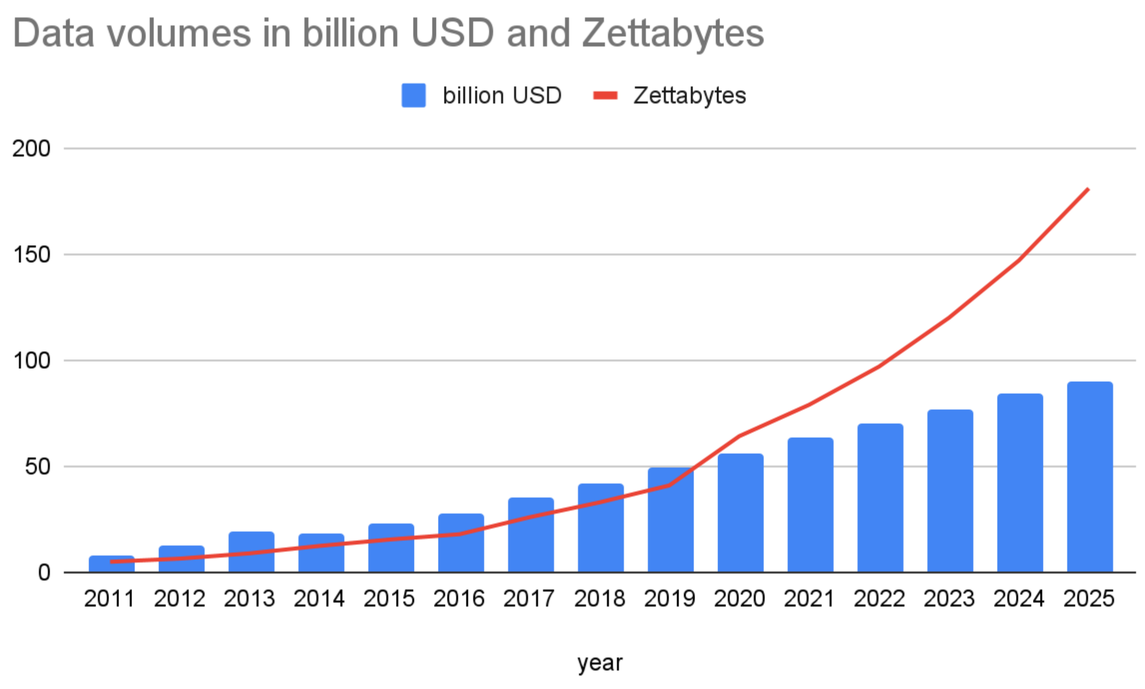


Figure 1-.2: This table shows the growth in data from 2011 with forecasts from 2019 to 2025. The table shows the growth in volumes both in billions of US and zettabytes. The data comes from IDC and Gartner.

With this growth in demand and increased importance to business it is arguably becoming increasingly more significant to manage, store and extract value from this data within business. Part of this trend within business is democratising access to data, this means getting access to the data to the decision makers. Visualisation tools such as Tableau and Microsoft PowerBi are becoming increasingly more common as data is becoming too large to manage within a desktop environment using spreadsheets.

Traditionally data storage systems have been heavily siloed data. The ability to search and extract data from a range of different systems has been compromised by the differing architectures that each repository uses. Even languages used to extract data from these data silos such as the Structure Query Language (SQL) often require a high degree of technical knowledge.

The initial idea behind this project was to increase the ease at which data could be extracted from a range of data silos. With this in mind developing a solution that could take a natural language statement and create a query capable of querying a data repository seemed to be the ideal solution. The problem was that these Natural Language Interface to a Database

(NLIDB) solutions have been within the research community since the US Navy's Ladder system in the 1970's.

Researching the use of NLIDB solutions it became clear that the number of solutions that had moved from the research community into general available solutions for use within the commercial market was not small. Most of the research projects looked at refining algorithms that could perform the translation from a natural language to a language capable of querying a database. Yet there seemed to be no description within the literature about how these solutions could be implemented or what the interface was between the non-technical user and the data silo was. As can be seen in Chapter 2 there also seemed to be a gap between expectation and reality from the solutions that had been implemented. This gap existed in terms of the performance of implemented systems. Researchers commented that the accuracy of the results returned from NLIDB solutions was not accurate enough to provide a truly useful solution. There also appeared to be a performance issue when it came to the speed of the implemented solution.

With these three facts this project looks at trying to provide a common interface to the implementation question of how can the algorithms that have been implemented by a range of researchers be accessed by non-technical users, and how can the performance in terms of accuracy and speed be improved to make these systems viable to the non-technical user. With this in mind the project looks to create its own domain specific language based on natural language using Bloom's methodology which is discussed in Chapter 6. The project also looks to simplify the process of converting a natural language into a language capable of querying a data repository. This simplification which is discussed in Chapters 4 and 5 looks to increase the speed and accuracy of the conversion process.

1.2. Contribution

This thesis contributes to two areas in the field of natural language interface into a database.

Domain Specific Language: This project introduces domain specific languages to the problem of converting a natural language into a language capable of querying a data repository. The Domain Specific Language within this project is used to provide an interface between the user, someone creating the natural language query and the data repository to be queried. Current projects provide no clear interface between the user and the repository but concentrate purely on the conversion from natural language to a structured query language.

Simplified Algorithm: This project also shows how simplifying the approach used to perform the conversion can elicit performance within both the speed and accuracy of conversion. This project shows that using shallow parsing can provide an accurate conversion. It also highlights the use of a sequence-to-sequence model can be useful for not just converting one natural language to another but also be used to convert from a natural language to a machine language capable of querying a database.

The use of domain specific languages (DSL) within this field has not been covered extensively within this field. The use of MELT within the GCC compiler as proposed by Starynkevitch (2011) and Polosukhin et al (2018) introduces a DSL and SpeakQL from Shah et al (2020) provide some insight into how a DSL could be used. However, none of these projects provide much detail of the DSL being used. Both MELT and SpeakSQL provide limited functionality and do not appear to be extensible. The DSL being proposed in Chapter 7 is being proposed as extensible and unlike other papers reviewed in this thesis it provides details as to how the DSL works and it describes an interface into how an algorithm that is implemented in Chapter 6.

This project shows how Part of Speech can be used to extract keywords from the input natural language statement that can be used to identify database tables and table columns. Taking this approach removes the ability of the system to handle language nuance but does identify the key elements of the natural language statement that can be used in the conversion to an SQL statement.

1.3. Summary

The thesis is organised into the following sections:

Chapter 2: Provides a literature review of the existing concepts which are used within this project relating to the simplification of the conversion process from natural language to a language capable of querying a data repository. The topics within this chapter are looking at the development of solutions based on semantic parsing, simplified learning model and augmented memory. It also looks at the use of sequence-to-sequence models, neural encoder decoder and multi stepped solutions. This chapter also highlights the novel approach that is being taken by this project.

Chapter 3: This chapter provides an overview of the architecture used by this project. This chapter also introduces the concept of a domain specific language as an interface between the user, someone creating the natural language query and the data repository to be queried. It also introduces a simplified algorithm and shows how simplifying the approach used to perform the conversion can elicit performance within both the speed and accuracy of conversion. This project shows that using shallow parsing can provide an accurate conversion. It also highlights the use of a sequence-to-sequence model can be useful for not just converting one natural language to another but also be used to convert from a natural language to a machine language capable of querying a database.

Chapter 4: The discussion in this chapter looks at the use of a domain specific language in the development of an NLIDB solution. As can be seen from this chapter the use of a DSL within the translation of a natural language to a language capable of querying a database has not been implemented and the discussion around the topic is limited to the use of an internal DSL as part of the conversion process. None of the papers reviewed for this thesis provide details as to how the proposed algorithm can be implemented. This chapter looks at the development of a domain specific language for providing an interface into the conversion algorithms developed in Chapters 5 and 6. Using Bloom's taxonomy a more natural domain specific language is created, which is far easier to learn than SQL. This chapter also introduces how the algorithms created in Chapter 5 and 6 can be used to underpin the DSL.

Chapter 5: This chapter proposes dismissing the use of language nuance as part of an NLIDB solution. The argument being that language nuance is not present in an SQL statement and therefore should not be used when parsing from natural language. Instead, this chapter proposes using part of speech (shallow parsing) as part of an algorithm that can be used to create a statement capable of being used to query an algorithm. The algorithm proposed in this chapter introduces a novel approach to parsing with the aim of improving the speed and accuracy of the conversion.

Chapter 6: This chapter takes the work completed in chapter 4 and extends the proposed algorithm with a novel approach to parsing the natural language input statement. This chapter proposes that SQL is just another language with its own syntax and grammar. The idea proposed in this chapter combines the use of sequence-to-sequence models in conjunction with the natural language part of speech technologies and domain specific languages to convert natural language queries into SQL. The approach being proposed by this chapter is to use natural language processing to perform an initial shallow pass of the incoming query and then use

Google's Tensor Flow to refine the query with the use of a sequence-to-sequence model. The thesis is also proposing to use a Domain Specific Language (DSL) as part of the conversion process. The use of the DSL has the potential to allow the natural language query to be translated into more than just an SQL statement, but any query language such as NoSQL or XQuery.

2. Related Work

2.1. Introduction

Research within the field of natural language to database compliant query has been ongoing for a number of years. Early systems like LADDER from the 1970 were developed by the US navy. The goal of such systems has been to create a simplified user interface so that users could query a database without the in-depth technical knowledge normally associated with database queries. The ever-pervasive growth of data available to users and consumers has kept this field of research relevant. This chapter looks at the current research being carried out in the area of natural language to structured query language (SQL) from 2016. The reasoning behind picking this time period is that in 2016 it appears that the research in NLIDB solutions changed. This change is highlighted by the review carried out by Wang et al (2016) and Allamanis et al (2018). Wang et (2016) highlights the increase in the number of machine learning projects used within the field of NLIDB. The Allamanis et al (2018) review also highlights that research in the field of NLIDB expands into other fields of research.

During the period of 2016 to 2020 a few reviews of the use of natural language to sql projects were carried out. In a review performed by Wang et al (2016) their work concentrated on projects that focused on the use of machine learning techniques. According to the review by Wang et al (2016) they categorise deep learning models into three groups according to the types of connections between layers. The review identified feedforward models (direct connection), energy models (undirected connection) and recurrent neural networks (recurrent connection). In contrast to Wang et al (2016) paper the later review carried out by Allamanis et al (2018) looks more into the use of probabilistic models in interpreting natural language statements into programming languages in general rather than just a language capable of querying a database. Their work also looks at the implementation of these models within the research arena.

The more recent review carried out by Kalajdjieski et al (2020) does not just look at the methods used to convert natural language to SQL. In contrast to the reviews carried out by Wang et al (2016) and Allamanis et al (2018) it also includes a review of the datasets such as the *WIKISQL* dataset created by Zhong et al (2017) which can be used to test and validate a model's performance. The work also includes a section on the models used to evaluate performance of natural language to SQL. This review has identified that since 2018 most of the work in the area of NLIDB appears to be in the areas of sequence-to-sequence models and semantic parsing. There is also a growing body of work in the area of augmented memory and neural encoder decoder for use in NLIDB systems. A number of other approaches have also been identified by this chapter that are being used to improve the translation of natural to SQL. These approaches have been categorised into logical groups for this chapter and they are highlighted below. AI solutions being proposed by this project is described in more detail in chapter 6. The concept behind it is to use a sequence-to-sequence models as part of the conversion from natural language to a language capable of querying a database. The sequence-to-sequence model uses LSTM to improve the accuracy of the conversion.

2.2. Identified Approaches

This review has identified a number of common approaches that have been used within the research community to improve the performance of natural language conversion to structured query language. Papers reviewed in this section have been grouped together based on the commonality of their research and the following sections highlight the common threads. Most of the papers reviewed concentrated on a single aspect of the conversion from natural language to a language capable of querying a database. The section on multi-step architecture looks at papers that combine several approaches to achieve the improvements in converting natural language to SQL. This multi-step approach is similar in principle to the approach being proposed by this project. As will be discussed later in the project the proposal is to combine shallow parsing (chapter 4) with a sequence-to-sequence model (chapter 5) and to also provide a domain specific language (chapter 6).

2.2.1. Semantic Parsing

The concept of semantic parsing in its simplest form is taking a natural language statement and converting it to a logical form that is machine understandable. Lin et al (2018) take this concept in its purest form to convert natural language to bash. Lin et al (2018) also highlights that it is not just SQL being used as a target for converting natural language to machine capable language. This chapter concentrates on the use of SQL as a target language. The research by Shah et al (2020) is based on speech to SQL and as part of their solution they propose a new language SpeakQL. They also create a dataset specifically for speech-based SQL conversions. The conversion process from speech to SQL the research still uses techniques used for typed natural language conversion. The approach used by Shah et al (2020) relies on semantic parsing for the creation of the SQL statement.

Though the use of semantic parsing four approaches seemed to be favoured by researchers in the goal of improving natural language to SQL conversion. These approaches can be classified under the following headings: executed guidance, tree structures, underlying database structure, descriptive language and user interactions.

The first of these executed guidance uses statistical analysis to select the best output from a number of possible solutions. This approach originates from the work carried out by Wang et al (2018b). Their concept looks at statements in various stages during the conversion process and discards those statements that cannot complete the conversion to SQL. Yin, Neubig (2019) take a similar approach to Wang by ranking the predicted output from the conversion model selecting those with the highest score. Talmor and Berent (2018) use the internet as their model for training.

The equation defined by Yin & Neubig (2019) summarises the work also carried out by both Wang et al (2019b), and Talmor and Berent (2018). The equation highlights the use of $s_t$as a bookmark to keep track of the generation process at each step (*t*) within the LSTM cycle.

$$s_t = f_{LSTM}\ ([\alpha_{t-1} : \underline{s_{t-1}} : \rho_t\ ], s_{t-1})$$

$\alpha_{t-1}$is the embedding from the previous step while $\underline{s}_{t-1}$ is the input into the vectorρ_t .

The use of tree structures for solving the problem with semantic parsing has been used by both Cheng et al (2018) and Yin et al (2018). In the case of Cheng their work uses the tree structure with a domain grammar to ensure that the conversion is accurate. In contrast Yin et al use tree structures to hold the training data which can be labelled or unlabelled.

Karki et al (2019) and Bogin et al (2019) both rely on the underlying database structure as part of the process in parsing the natural language statement. Bogin et al (2019) model the database structure within a graph schema as a method of understanding the relationships between tables. In contrast Karki et al (2019) construct a row and column-based grid from the database features.

The use of a descriptive language can also be used with the semantic parser. In the case of Yin and Neubig (2018) they propose using abstract syntax description language for parsing the natural language onto an SQL template. Lin et al (2019) use a schema dependent grammar to map the natural language onto a SQL syntax. Campagna et al (2019) look at using a Virtual Assistant Programming Language (VAPL) to formalise the natural language statement. Cheng et al (2019) take a similar approach to Campagna et al (2019) as they use templates that can map the text from the natural language onto an SQL structured template.

Other approaches that rely on semantic parsing such as Yao et al (2019) rely on user interaction to improve the performance of the conversion by posing questions to reduce ambiguities. Whereas Benajiba et al (2019) explore the possibilities of Semantic Pattern Similarity (SPS). This approach compares the structure of sentences with known structures and then evaluates that to the SQL query. Zhang et al (2020) use a sketch-based approach to semantic parsing. Their proposed approach does not convert every element of the SQL but extracts just the values and the columns. This is very similar to the approach being taken by this project, but instead of sketching this chapter uses a sequence-to-sequence model.

2.2.2. Sequence to Sequence Models

In contrast to semantic parsing the concept of a sequence-to-sequence model is to train a model to take a sequence from one domain or language and convert to another domain or language. These models have been used to convert text from one natural language to another and have now been introduced to convert from a natural language to a programming language such as *bash* in the case of Lin et al (2018) or *SQL* as proposed by Shi et al 2018.

Recent research using sequence to sequence models has extended the approach typically used by Lin et al (2018) which takes a natural language statement and compares it to a *bash* command. Sequence to sequence research projects can be categorised into a number of streams those that have begun using the content from the database rather than the database structure to understand the structure of the data, those that are chunking the natural language statement into smaller more discrete blocks to create multiple seq2seq models. There are those using a scoring mechanism to rank the decoder as well as using the underlying database structure. There is also work on creating sequence to SQL models.

In the model proposed by Shi et al 2018 the work uses a sequence to action model. Their solution uses a SQL template with place holders to contain the name of the table, columns and variables. The sequence to action model is then used to parse the appropriate values into the template. With this project the underlying database structure is central to the conversion process.

There are also extensions to the traditional sequence to sequence model such as the work carried out by Xu et al (2018) which uses a graph based neural network to create a graph to sequence model. Then the work carried out by Yu et al (2018) uses a tree network to create what they refer to as a text-to-SQL model. Wang, Tian et al (2018) also take a similar approach with their text to SQL model. Wang instead proposes separating the data from the schema within the sequence model. Guo et al (2019) also propose a text-to-SQL model by creating a multiple step approach to the problem of converting natural language to SQL. As part of the process the solution creates a synthetic query language from the natural language statement and database structure. The final query is inferred from the synthetic query. The figure summarises the approach taken by Guo et al. Within the natural language encoder *x* represents the natural language statement chunked into individual word tokens.

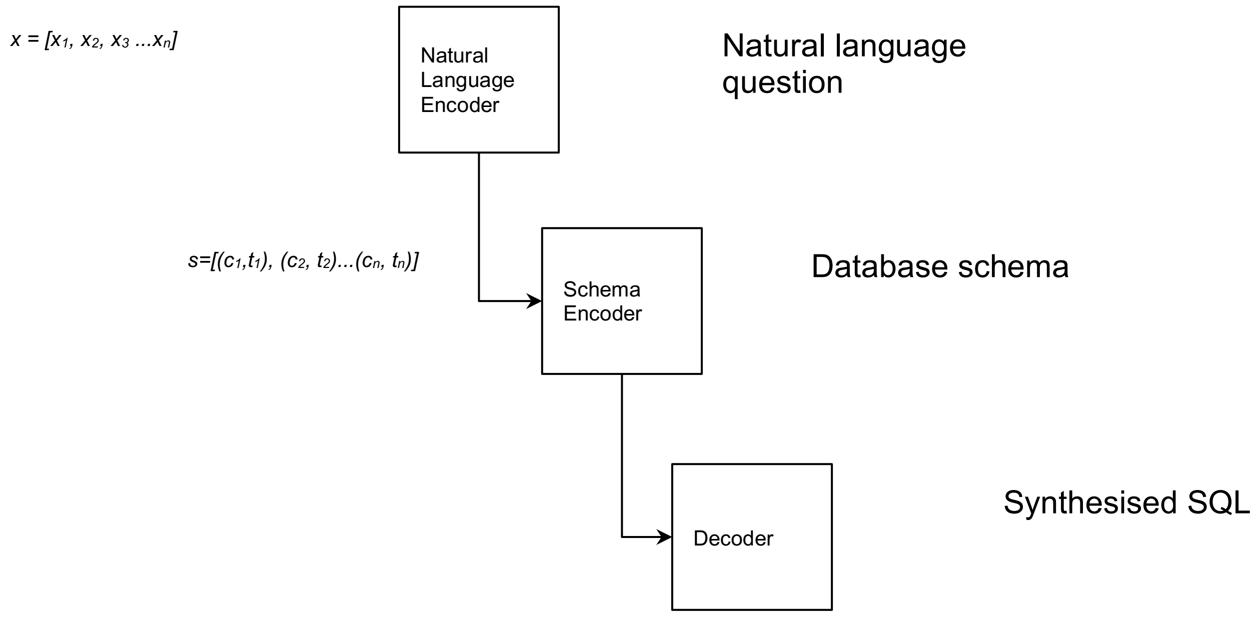

$x = [x_1, x_2, x_3 \ldots x_n]$
Natural Language Encoder
Natural language question
$s=[(c_1,t_1), (c_2, t_2)\ldots(c_n, t_n)]$
Schema Encoder
Database schema
Decoder
Synthesised SQL

Figure 2-.1: summarises the approach taken by Gua et al (2019)

For the schema encoder *s* represents the target database schema with C representing the database columns and t the database tables of the target database. The output from the decoder is a synthesised SQL statement.

In parallel to the extension of the sequence-to-sequence model the more traditional sequence to sequence model is being refined by the likes of Soru et al (2018) which like Xu et al (2018) uses graph patterns to learn the sequence make up relationships between elements. The work by Soru though is more of a traditional sequence to sequence approach. In comparison Su et al (2018) propose using multiple sequence to sequence models at each step along the process of conversion. They also support the use of user interaction to correct errors in the process of converting natural language to SQL.

Part of a sequence-to-sequence solution relies on the decoding or the translation from the input to the output. Bello et al (2018) assigns an item score as part of the decoding process. The score is based on historical data, and according to Bello allows for *higher-order interactions*. In contrast Zavershynskyi et al 2018 use a multiplicative attention mechanism as part of the RNN within the decoder.

The work performed by Guo and Gao (2018) like Su et al (2018) chunked the natural language statement into smaller elements thereby creating a chain of sequence-to-sequence models. Within the *WHERE* clause the team ranked possible solutions based on historic data choice the highest rank.

Other approaches for handling sequence to sequence models like Petrovski et al (2018). The team proposes removing the database structure completely from the sequence-to-sequence model and relying solely on the content of the tables to describe the content of the database table. Then Sabour et al (2019) were more concerned with the method of training the sequence-to-sequence model. Their approach was to create an *Optimal Completion Distillation (OCD)*. This required statistically sampling the data used for training based on predefined characteristics.

2.2.3. Simplified Learning Model

Within machine learning a large corpus of data is traditionally used to train a model. The process is exactly the same as that used by sequence-to-sequence models yet the biggest difference in using the simplified learning model is the amount of data used within the training model. The concept of a simplified learning model is to use a much smaller dataset for training. The work of Huang et al 2018 identifies that such a broad-brush approach to training can be problematic. Their approach was to reduce the amount of training required by the model by specifying which simplified task the machine learning model should concentrate on. In the case of Huang et al 2018 the model should use the metadata of the underlying database for training.

Yavuz et al 2018 simplified the problem and had the sole purpose of trying to score 100% in translating the natural language queries from the Zhong et al 2107 corpus into SQL. The solution proposed takes the question and the database table from the corpus and concentrates on generating the *WHERE* clause. The solution uses a *neural network bi-directional LSTM for each word in the question*.

2.2.4. Neural Encoder Decoder

A neural encoder decoder architecture is a neural network design pattern that takes an input and produces an output. The formula proposed by Shin (2019) shown below neatly highlights the encoding of the natural language input statement. The Bidirectional long short-term memory encoding takes the tokenised elements from the input statement and without sharing results across the three algorithms identifies whether the tokens belong to either the question, table or column.

$$(c_{i,0}^{fwd}, c_{i,0}^{rev}), \ldots (c_{i,|c_i|}^{fwd}, c_{i,|c_i|}^{rev}) = BiLSTM_{column}(C_i^{type}, c_{i,1}, \ldots, c_{i,|c_i|}); c_i^{init} = Concat(c_{i,|c_i|}^{fwd}, c_{i,0}^{rev})$$

$$(t_{i,0}^{fwd}, t_{i,0}^{rev}), \ldots (t_{i,|t_i|}^{fwd}, t_{i,|t_i|}^{rev}) = BiLSTM_{table}(t_{i,1} \ldots, t_{i,|t_i|}); t_i^{init} = Concat(t_{i,|c_i|}^{fwd}, t_{i,1}^{rev})$$

$$(q_i^{fwd}, q_i^{rev}), \ldots (q_{|q|}^{fwd}, q_{|q|}^{rev}) = BiLSTM_{question}(q_1, \ldots, q_{|q|}); q_i^{init} = Concat(q_i^{fwd}, q_i^{rev})$$

The equation is split into three LSTM functions, one for the question where each word or token from the input is represented by '*q'.* The second LSTM identifies the table from the tokenized input statement where *'t'* represents the table. The third LSTM is used to identify the column from the database represented by *'c'* that is extracted from the input tokens. The decoder just builds up the resultant query from the most likely results of the encoder. The BiLSTM function performs a look up on each word in the input question. Additionally, each of the LSTM from the equation do not share any parameters

There are a number of similarities between this approach and sequence to sequence models. Though sequence to sequence models do use neural encoders and decoders the approach used by the papers discussed in this section is not to emulate a sequence-to-sequence model but to use the neural encoder decoder to understand the relationship between either the natural language question and the subsequent SQL or the relationships between tables in the database.

The work by Shin (2019) is concerned about the relationships between tables within a database structure. Their paper uses a graph database to handle the relationships between the tables. This approach appears to have a good performance when working with a natural language query that spans multiple tables but appears not to have been tested against a single table. There is also no information as to the upper limit on the number of tables that this approach can handle.

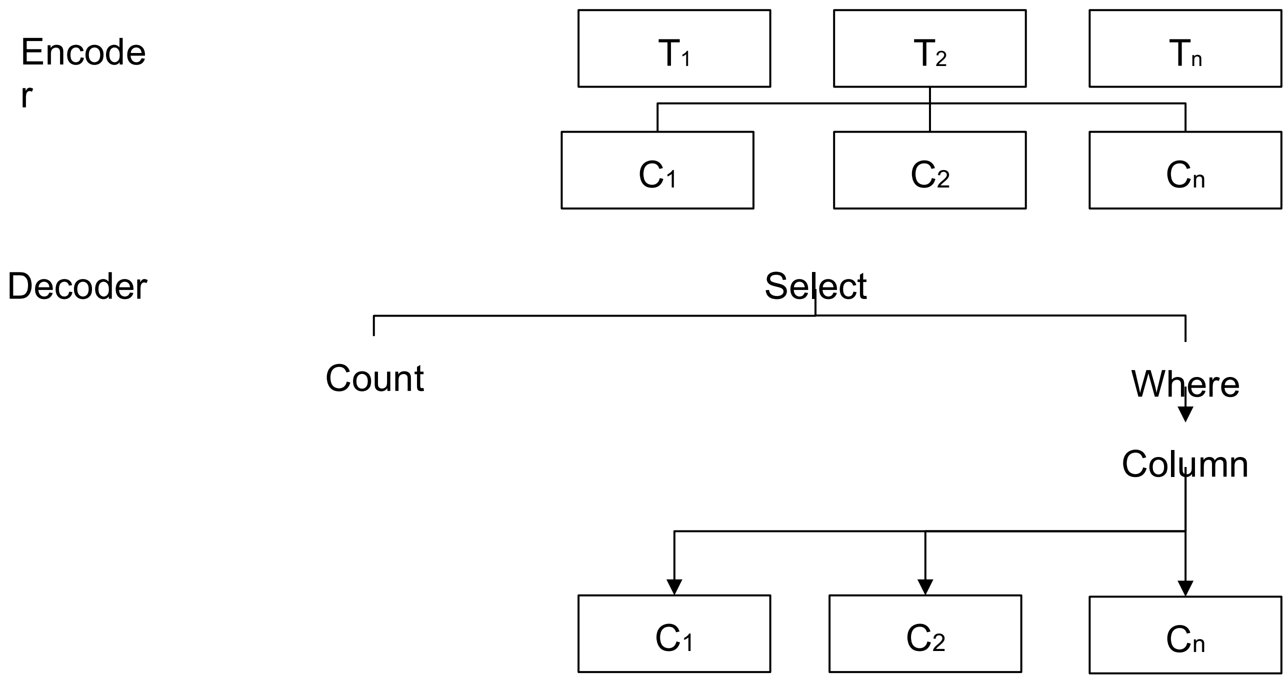


Figure 2-.2: The figure shows an overview of the encoder and decoder architecture typically used within a neural encoder decoder solution. The diagram comes from the work proposed by Shin (2019). T represents each table in the underlying database and C represents the columns within the tables.

In contrast to the approach taken by Shin (2019), Cho et al (2018) use a multi-layer sequential network with attention supervision. The model is trained against a value pairing of questions and answers. The paper makes no account of the underlying structure of the database. It would appear that creating a training set for the approach proposed by Cho et al (2018) would require a high degree of manual intervention.

Chen et al (2019) identified a number of inefficiencies in the traditional encoder decoder paradigm. Their solution uses a multiple step approach with a series of intermediate steps that are then processed as part of a chained series of events.

The work carried out by Suhr et al (2018) and Wang et al (2018) take very similar approaches. Both projects use an encoder decoder model in conjunction with a sequence-to-sequence model. These papers also use the underlying syntax of the database as part of the translation process from natural language to SQL. Suhr also uses regular expressions as part of the process to build up the SQL query.

2.2.5. Augmented Memory

Augmented memory is a simple approach to leverage a memory buffer. This buffer contains a matrix of information used as part of the conversion process. In the case of Jai and Lang (2016) the augmented memory is used to inject prior knowledge into the RNN model used by the sequence-to-sequence model. McCann et al (2018) uses a similar approach first proposed by Jai and Lang (2016) when they create the Natural Language Decathlon (decaNLP). The decaNLP was created to test the NLP multi domain solution multi-pointer-generator decoder.

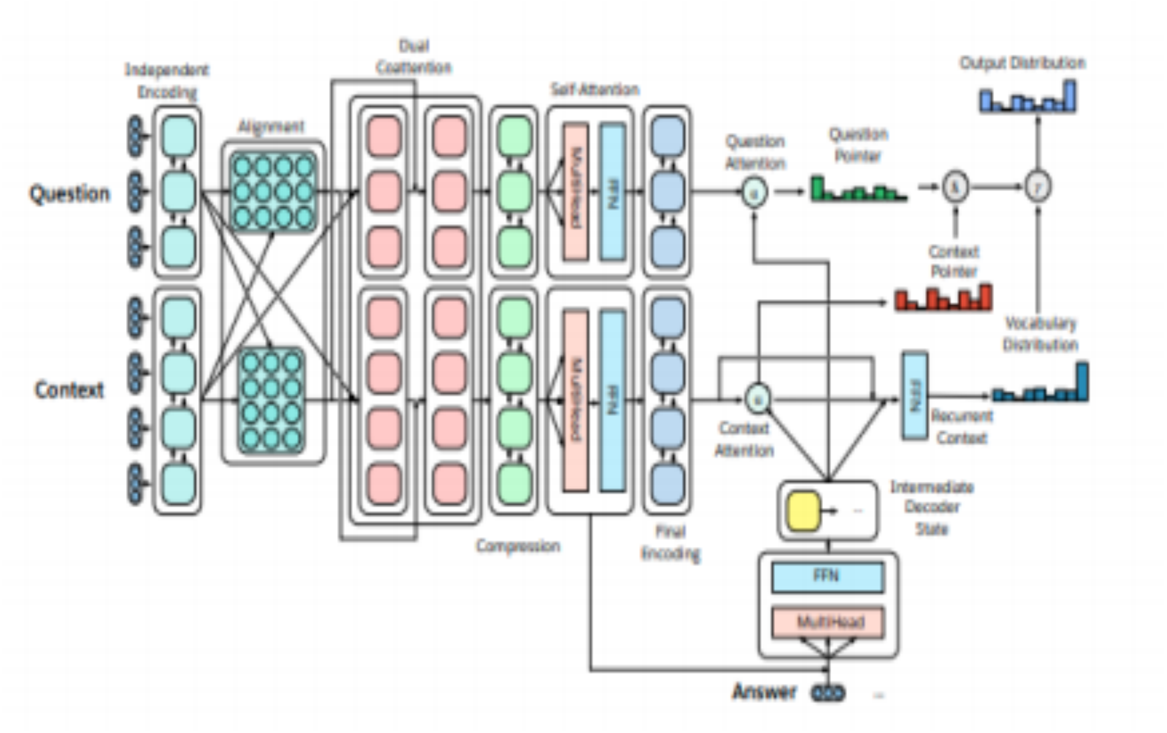


Figure 2-.3 From the McCann et al (2018) paper showing the architecture of their proposed solution.

This approach was first proposed by Jain and Lang (2016) and was expanded by a range of papers. The work by Dadashkarimi, Tatikonda (2019) uses a similar approach to McCann et al (2018) when they introduce the concept of a cache which stores the required vocabulary for use during decoding. The cache can also be used to find related tokens. Xiong and Sun (2019) use the augmented memory approach when target domain data is lacking. Liang et al (2018) in a similar manner to Xiong and Sun (2019) use the memory to speed up training.

2.2.6. Corpus Tagging

The work carried out by Zhong et al 2017 relied on a user community to manually tag over 80000 data points. With this tagged corpus the translation of natural language query to a language capable of querying a database could then begin. The use of natural language processing requires a tagged corpus, whether that is tagging each word within a construct with either a label to identify the work as a noun, verb adjective or an asset in a database such as a table, column or variable. The equation that summaries the work of Zhong et al is:

$$x = [\text{<col>}; x^c_1; x^c_2; ...; x^c_N ; \text{<sql>}; x^s; \text{<question>}; x^q]$$

The equation denotes *X* as a token from the input statement *<question>* as a relation to the columns *<col>* in the underlying database and an attribute of the SQL statement *<sql>*. *X* is then passed through a bi-directional long short-term memory network. The key to the work proposed by Zhong et al (2017) is the tagging of the large corpus of training data that facilities the accuracy of the proposed solution and also provides the research community with a large corpus of tagged data know as the ‘WikiSQL’ dataset.

Work is being carried out to refine and automate the process of tagging natural language corpus. Such work was carried out by Shah, Tur and Tur in 2018 with the sole purpose of using machine learning to create a tagged dataset that can be used for a NLIDB system.

Finegan-Dollak et al 2018 highlight the accuracy issue with tagging large datasets by identifying errors in the tagging across multiple datasets and correcting those errors. With this work they also identify that tagging a corpus requires being able to identify a range of properties that are not easily identifiable.

The work by Weir et al (2020) in contrast to the other papers reviewed in this section creates synthetic data to train the model. The work proposes using existing data as a basis for the synthetic data. Wier et al (2020) argue that the synthetic data provides a more accurate and smaller more concise dataset for training, thereby reducing training times.

2.2.7. Multi Step Architecture

Most approaches that have been discussed within this chapter have concentrated on refining a single component to improve the performance of a NLIDB solution. Few have taken a multi-step approach using multiple technologies within an overall solution. Polosukhinet et al (2018) use a domain specific language (DSL) along with an extension to the sequence-to-sequence model that they refer to as Seq2Tree. This work is similar to Shi et al 2018 but Polosukhinet decides to use a DSL rather than ad-hoc regular expression.

Likewise Lukovnikov et al 2018 use a combination of augmented pointer along with LSTM *column encoders*, along with a sequence to sequence model in conjunction with semantic parsing to translate the natural language statement into a query language. Taking a similar approach is Choi et al (2020) again using sketching to extract the pertinent data from the natural language input statement which can then be transposed on the SQL template. They also propose recursively predicting nested statements.

The work by Joshi et al (2020) takes a hybrid approach to the conversion of text to SQL. Their work uses a series of sequence-to-sequence models to create the SQL statement they also propose user interaction. Unlike Gur et al 2018 who propose user interaction to refine the process, Joshi et al (2020) are after restricting the inputs from users to avoid linguistic variations and ambiguities in the statement.

2.2.8. Other Approaches

Apart from the main topics of research that have been identified and discussed in this literature review there are several smaller nuanced research projects that have been identified. These projects are more embryonic in nature and usually have a single paper dedicated to the subject. The research discussed in these papers has not been followed up with more papers, yet the approach they propose could be pertinent to the task of converting natural language to a structured query language. The list below highlights these approaches.

Meta data from the underlying database has been used to support and refine the process of conversion from natural language to a database query language. The work proposed by Agarwal et al 2019 uses this metadata to train a machine learning model to refine the translation from natural language to SQL.

Slot filling as proposed by Yu et al (2018) uses a slot filling regex approach to mapping words in the language statement to database assets. The work uses techniques similar to domain specific languages by building up a structure of the SQL query and then using regular expressions fills in the gaps in the language structure with data from the natural query.

SQL log approach proposed by Baik et al 2019 leveraging information database log file to refine the performance of the conversion of natural language to database query. Like the work carried out by Agarwal et al 2019 the approach uses information within the database architecture to refine the process.

Subjective Databases was introduced by the work undertaken by Li et al 2019. The concept behind this approach is the meaning of contextual data from the underlying data. The approach has some similarity to that employed by sequence-to-sequence models. From this work came the OpineDB subjective database system.

User interactions The work carried out by Gur et al 2018 proposes using interaction from the user to refine the process of conversion. The researchers identified that the *where* clause had the highest potential to be erroneous. To combat this perceived weakness users are asked multiple choice questions to validate or correct errors within the conversion.

Neural classifier is proposed by the work carried out by Wang et al (2020). The classifier detects specific components in the natural language. This is used with the underlying database structure and a sequence-to-sequence model to translate the natural language to SQL.

Ellis et al 2018 We introduce a model that learns to convert simple hand drawings into graphics programs written in a subset of LATEX. The model combines techniques from deep learning and program synthesis., We developed a deep network architecture for efficiently inferring a spec, S, from a hand-drawn image,

Augmented Pointer The Augmented Pointer is used in conjunction with a sequence to sequence model for converting natural language to SQL. The approach first proposed by Vinyals et al (2015) uses a bidirectional LSTM network to encode the input statement. In the case of Zhong et al (2017) the input statement also contains a list of the database column names. The output from the LSTM network differentiates between the column names required for the returned dataset and the search conditions such as the WHERE clause.

2.3. Conclusion

Most of the work carried out within this field concentrates on a single approach such as the sequence to SQL approach proposed by Zhong et al (2017) or natural language to SQL proposed by Weir et al (2020) and Zhang et al (2020). There is also a body of work that takes a number of approaches such as Joshi et al (2020) and Polosukhinet et al (2018). There has also been the development of innovative approaches such as Shin (2019) which has seen the development of research into Neural Encoder Decoder which has come from the work into sequence-to-sequence models. Development of deep learning models has led to the introduction of novel approaches that have been used by Wang et al (2016) and Ellis et al (2018).

The semantic parsing approach takes the natural language input statement and tries to map the natural language words or tokens into a structured query language structure. The approach often follows a slot filling Methodology and has a certain similarity to the approach proposed by Sequence-to-Sequence models. The approach proposed by Seq2Seq explicitly converts a word of phrase from one state to another through the use of a mapping file. The domain for sequence-to-sequence models seems to have originated in natural language to natural language translation (ie Spanish to English). The work undertaken by the likes of Lin et al (2018) and Shi et al (2018) have extended the use of sequence-to-sequence models to include computer based languages such as natural language to bash.

The simplified learning model approach and the neural encoder decoder have certain similarities. Both approaches use neural networks as part of the translation from natural language to structured query language. The idea behind the use of the simplified learning model is to use a smaller dataset and reduce the amount of time required to train the model. In contrast, neural encoder decoder are like sequence to sequence models more concerned about modelling the relationship between the natural language input and the resulting query language output.

Though this chapter categorised the research topics into six distinct groupings the equation that summaries the work of Zhong et al is:

$$x = [<col>; x^{c}_{1}; x^{c}_{2}; ...; x^{c}_{N} ; <sql>; x^{s}; <question>; x^{q}]$$

The equation denotes *X* as a token from the input statement as a relation to the columns in the underlying database and an attribute of the SQL statement. *X* is then passed through a bi-directional long short-term memory network. The key to the work proposed by Zhong et al (2017) is the tagging of the large corpus of training data that facilities the accuracy of the proposed solution and also provides the research community with a large corpus of tagged data know as the ‘WikiSQL’ dataset. There is a blurring of the lines between the topics as a certain amount of cross over exists across each topic. The topics that have been identified in the multistep architecture highlight the interconnectivity between the project, as in the case of Lukovnikov et al 2018 which use both augmented pointer and sequence to sequence models.

Of all the papers reviewed only Shah et al (2020), Agarwal et al (2019), Chen et al (2019), Guo et al (2019), Liang et al (2018), Polosukhin et al (2018) and Xiong et al (2019) discuss the possibility of using or creating a Domain Specific Language as part of the solution. With Shah et al the domain specific language they propose is SpeakQL, which takes spoken text and converts to SQL and Guo et al (2019) have devised the DSL SemQL which is proposed as an intermediary between natural language and SQL. Polosukhin et al (2018) mention the possibility of creating rich domain specific language but provide no details as to how nor mentions that one has been developed. The rest of the papers reviewed for this project provide no detail about how to interact with the changes to the algorithm being proposed. All the research reviewed by this chapter could have been incorporated into a domain specific language, yet they do not mention nor propose how.

The concepts that the papers in this review have concentrated on are to provide alternative methods in converting a natural language statement into a language capable of querying a repository. Details as to how each of these projects improves the speed and accuracy of that conversion process are light on detail in many papers. Also from this review it appears that there is a gap in the research which could channel all of these projects into an extensible domain specific language used to convert natural language into a language capable of querying a repository. It is possible that the complexity of creating a DSL was outside the scope of these projects, which was focused purely on refining the conversion from natural language to a language capable of querying a database. This project will look at the use of a DSL as an interface to the conversion process, as well as trying to improve the performance of conversion.

3. Architectural Overview

3.1. Introduction

With the increasingly pervasive nature of data and its ever expanding use to answer questions within business science, medicine extracting data is becoming more important. The idea behind this project is to make data more democratised by allowing access to it without the need for specialist languages.

This thesis proposes a solution to solve both the language nuance highlighted by Florin et al (2017) and Kiev et at (2011) as well as the performance issues with the use of shallow parsing as discussed by Joshi and Akerkar (2008). The use of shallow parsing, also referred to as part of speech, negates the requirement for an understanding of language nuances, as key words are extracted from the input statement and used within the conversion process. The shallow parsing approach being proposed by this chapter is the use of keywords. This approach first proposed by the Ratnaparkhi (1996) is used to identify characteristics of the input statement that are important for the search. In contrast to Ratnaparki's approach this project not only identifies the keywords that would be useful in the translation process, but maps the keywords to tables and columns within the tables. This high level overview of the proposed solution hopes to provide an approach that can improve the accuracy and speed of conversion.

This chapter will introduce the use of an index file containing keywords extracted from the underlying database that identify the tables and associated columns. It is this approach that helps build the performance in translation from natural language to SQL. This builds on the work of Jwalapuram & Mamidi (2017) who are among a number of authors who have carried out research into using keywords to enable NLIDB based systems to perform searches. Unlike that used by Jwalapuram and Mamidi (2017) this project uses Part of Speech (POS) processing in conjunction with an index file which allows for individual words to be extracted from the natural language query. The individually extracted words can then be used to create the query for the NLIDB solution. Details of the architecture for the proposed solution can be found in section 4.4.

Having defined that accuracy of conversion and performance of conversion from natural language to a language capable of querying a database. The problem that this project is aiming to solve is to improve the speed and accuracy of that conversion process. Having also highlighted the work that is currently being undertaken within this arena. The next step is to introduce the solution being proposed by this project.

The work being carried out by this project looks at taking a new approach to the problem of converting natural language into language capable of querying a repository. It introduces the concept of a domain specific language as an entry or interface between the user and the algorithm. It may seem counterintuitive to replace the domain specific language SQL with another. Introducing the new DSL which is based on the Bloom Knowledge Taxonomy makes the interface for the user far simpler than using SQL. Bloom’s taxonomy is taught in school and is used extensively outside the computer industry.

This project also proposes a novel approach around the use of shallow parsing as part of the conversion process. The idea being argued by this project is that trying to understand language nuance does not aid the accuracy of the conversion process.

Finally, this project introduces the concept of sequence to sequence models which are usually reserved for chatbots or converting one natural language into another. This novel implementation of sequence-to-sequence models to convert natural language into a computer based language. An internal DSL is also discussed potentially working with the sequence-to-sequence model to improve the speed and performance of the conversion process but is not implemented. It is however discussed as a future enhancement to the project.

3.2. Research Challenges

In the course of this project a number of challenges were identified within the key components. The key challenges to the solution of converting a natural language statement into a language that is capable of querying a database have frequently been described as accuracy of conversion and speed of conversion. For the accuracy of the conversion the aim is to return results from the database query that are expected. The question is how can this expected value be determined and how to measure the returned result against the expected result in that the results returned from the process match the request for information from the input statement. This measure of accuracy is examined in chapter 5 and chapter 6.

For the second key component, which is the speed of conversion. The problem was to return results from the underlying database without increasing the latency within the overall solution when compared to extracting data directly out of a database. In the age of search engines users have become accustomed to expecting data returned within a 5 second window.

- **Domain Specific Language:** Using the DSL provides a common interface into the use of a parsing algorithm. Existing research concentrates on the mechanism of how a solution works rather than on how to implement a solution. The use of a DSL to solve this problem has been hinted at the most notably from the work of Polosukhinet et al (2018) which uses a domain specific language (DSL) as part of the conversion process. Yet there is no detail on how the DSL is implemented. The paper by Skeggs and Lauria also mention in passing the use of a DSL but again there is no detail on the implementation nor the use of the DSL. An additional problem exists with the introduction of the domain specific language. That is how to measure the performance of using the DSL against not using it.
- **Shallow Parsing:** The use of shallow parsing is not unique to this project. Ratnaparki's and Jwalapuram & Mamidi (2017) also propose using a shallow parsing approach. The difference between the approach proposed by this project and the work carried out by Jwalapuram and Mamidi (2017) is that the keywords extracted are mapped directly onto the underlying target database. The challenge here is to try and correctly identify words from the input statement and correlate them with tables and columns from the target database.
- **Sequence to Sequence Models:** Sequence to sequence models area currently used for converting one natural language to another, such as Spainish to English. They are also used within online chat engines. The problem this project faces to turn a natural langugae statement into a language capable of querying a database. Work by Lin et al (2018) takes a natural language statement and then transfers it to a *bash* command. These commands are not full-fledged *bash* programs. Understanding language nuance will not translate into a SQL command.

3.3. Solution Overview

The overall high level solution architecture is shown in the diagram Figure 3.1. This diagram shows all the components and the process flow used within the framework for converting natural language into a language capable of querying a database. The diagram also shows the flow of data through the system and the steps required to get from the input natural language query to a language capable of querying a repository.

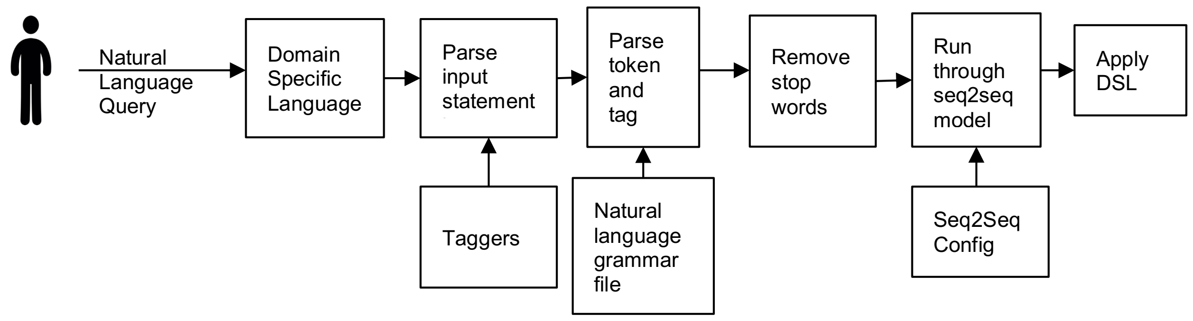


Figure 3-.1: The overall solution diagram shows the components that make up the complete solution.

The output from this system is a SQL query capable of querying a database. The simplified diagram in figure 3.1 highlights each step through the process of converting the natural language input into a language that is capable of querying a repository. Each step through the process is highlighted and an overview of the conversion process is highlighted. Individual steps in the process are highlighted below with more details of each step described in the following chapters.

- **Natural Language Query.** The natural language query is the input into the system, entered by the user and the starting point for the conversion process.

- **Domain Specific Language.** A DSL is being proposed by this project as an interface between the natural language query and the algorithm that converts the natural language into a structured query language. This project proposes using a DSL based on Bloom’s taxonomy for knowledge. The approach will define a set of guidelines that can simplify the conversion process as the structure of the natural language is of a defined structure. This structure is also not too restrictive ensuring users do not need specialist knowledge to use the language.

- **Parse Input Statement**, takes the incoming statement and basically chunks the statement into word tokens. For this project python NLTK taggers were used to identify words and produce the tags. This project chained a name tagger to identify names and a specially built parser to identify elements of the input string with known database attributes. Details of this process can be found in section 5.4. The output from this process is a list of names and words that can then be tagged.

- **Parse token and tag**, this tagging process works in concert with the parsers to identify real names as well as keywords from the tokens so that the correct NLP tag can be applied to the tokens. The tagging process takes the output from the parsing process and applies tags, which can identify names and elements from the underlying process. The output from this process is a JSON string with the chunked words from the input statement and the associated tags. The details of this step are highlighted in section 5.4.

- **Remove stop words,** stop words such as ‘the’, ‘a’, ‘is’ which have no assigned NLP tag are removed from the original query as they have no use in the conversion process. Removal of stop words is performed at this stage as identified stop words may have some meaning within the conversion process. The output from this step is the JSON string with the remaining tokenised words from the input statement and the tags associated with those words. Details of this process are shown in section 5.4.
- **Sequence to sequence model** takes the remaining word tokens from the input statement and proposes using a sequence-to-sequence model to convert the words into database elements such as table names and column names.
- **Apply DSL,** takes the output from the sequence-to-sequence model. The returned data is then mapped onto a template for an SQL query. Using an internal DSL within this step may provide more flexibility and allow the target language to be a language other than SQL.

The framework describes the steps that a natural language statement goes through to be processed into a query capable of querying a database. The entire process was refined over several iterations. Details of the model used by this project can be found detailed in chapters 5,6 and 7.

3.4. Conclusion

This project having highlighted gaps in the conversion of natural language to query language this section provides a high level overview of the potential solution that is being proposed by this project. This thesis contributes to three areas in the field of natural language interface into a database.

- **Domain Specific Language:** This project introduces domain specific languages to the problem of converting a natural language into a language capable of querying a data repository. The Domain Specific Language within this project is used to provide an interface between the user, someone creating the natural language query and the data repository to be queried. Current projects provide no clear interface between the user and the repository but concentrate purely on the conversion from natural language to a structured query language.

- **Simplified Algorithm:** This project also shows how simplifying the approach used to perform the conversion can elicit performance within both the speed and accuracy of conversion. This project shows that using shallow parsing can provide an accurate conversion.

- **Sequence to Sequence Model:** It also highlights the use of a sequence-to-sequence model can be useful for not just converting one natural language to another but also be used to convert from a natural language to a machine language capable of querying a database. Research is centred on using sequence to sequence models to translate one natural language to another such as Spanish into English. There is no detailed research on using sequence to sequence models to translate a natural language statement into a computer language such as SQL. A SQL statement is after all another language, like natural languages it has syntax, grammar and vocabulary. This project makes the proposal that sequence to sequence models can be used to translate a natural language into a computer language like SQL.

The next chapter will look at the use of domain specific languages in the field of NLIDB. Currently there is not a great deal of work in this field so the number of references are light. The discussions starts with a review of what is currently available.

4. Using a Domain Specific Language Within an NLIDB Solution

4.1. Introduction

Domain Specific Languages (DSL) are created to perform a very specific task within a narrow operating field and often to perform very specific tasks. Examples of DSL's are Structured Query Language for querying a database, and HTML used to display a web page in a browser. This chapter is going to introduce the concept of using a domain specific language to query a database based on a natural language query. The ultimate aim of this work is to create a DSL that is extensible thereby allowing the language to develop and expand thereby developing with user expectations. Currently the work being carried out in the field of natural language to database query looks to enhance a specific algorithm rather than providing a vessel that can be expanded encompassing these new thoughts, algorithmic and improvements to existing techniques.

The idea behind using the DSL is to provide an interface between the natural language query and the process to transform that query into a query capable of extracting data from a repository or database. Most papers in this field concentrate on an algorithm that can be applied to improve the conversion from a natural language into a language capable of querying a repository. There is little on how to implement the algorithm being proposed. The DSL provides the entry point to algorithms that perform the heavy lifting when transforming natural language queries to a language capable of querying a database. There is also a gap in the research field of NLIDB that the DSL can fill. The work carried out in the previous chapter looked at about 50 projects within this field and yet only Skeggs and Lauria (2018) as well as Polosukhinet et al (2018) mentions the use of a DSL. The paper from Skeggs and Lauria (2018) specifically suggests using the DSL as providing an interface between algorithms being proposed by the project and the end user. This thesis takes the DSL proposal and provides details on how to create and use it within a NLIDB solution.

Here we introduce an external natural query language that is capable of being used as part of a solution for a natural language interface to a database (NLIDB) solution. As shown in chapter 4 this approach has not been covered in detail by other research projects, and appears to be mentioned in passing by Skeggs and Lauria (2018) as well as Polosukhinet et al (2018). The advantage of using an external DSL is that the language becomes accessible to the user whereas an internal DSL would only be available to the conversion process. This project does recommend that an internal DSL is used in the final stages of the conversion process but it has not been implemented at this stage.

Attempts have been made in the past to create a natural query language such as the work by Sibuya et al (1978) when they created a query processing system called Yachimata. Their system concentrated on the Japanese language to create a query capable of querying a database. Their system uses a noun-phrase model to create a framework for creating queries. In contrast this project proposes to use strict language grammar to create a framework which will be used to transform the natural language query into a language capable of querying a database. Using the Bloom taxonomy for knowledge will provide the framework against which the rules for the DSL can be developed. The approach being proposed by this chapter will follow the recommendation of Rabiser et al (2018) and make the language extensible. The advantages being that the language is capable of developing as new algorithms and approaches develop.

Having designed and created the domain specific language the next step is to highlight the implementation of the language. This section will also look at how to implement algorithms into the language giving the language flexibility to adjust to changes in the research of creating a natural language interface to a database. The DSL adds a level of abstraction and provides a predefined structure based on the work of Bloom and the resulting taxonomy of knowledge. The use of this taxonomy will simplify the structure of the questions being asked by reducing the level of variation in the question and thereby simplifying the parsing of the incoming natural language query. The abstraction layer provided by the DSL enables algorithms to be injected into the DSL providing a constant interface between the user and the SQL statement as well as the flexibility to choose the underlying algorithm to perform the parsing task as well as the conversion to the resulting SQL statement used for querying the target data repository.

4.2. Related Work

Domain specific languages (DSL), as highlighted by the work of Klint et al (2009) and Starynkevitch (2011) can be used to simplify a specific task. In the case of Starynkevitch (2011), the work proposes using MELT a domain specific language in the GCC compiler to create plugins for the GCC Compiler. Starynkevitch (2011) argues that using the DSL reduces the complexity of creating plugins for the GCC compiler.

Like the work proposed by Starynkevitch (2011), Klint et al (2009) uses RASCAL, a domain specific language to simplify the process of analysing source code. The review of DSL's carried out by Rodrigues et al (2017) also reiterates the idea that DSL's by stating that one of the aims of a DSL is to *ease the role of developers*.

Langlois et al (2007) highlight the importance of design patterns in the development of a DSL, but also recognise that the number of design patterns in use were few. Their work describes a number features that a DSL should have

- **Language features** are formalised by abstract and concrete syntaxes. According to Langlois et al (2007) the concrete syntax describes language structure in human usable form. The abstract syntax characterises elements of the domain.
- **Transformation features** describe the flow of information through the DSL from the problem to the solution.
- **Tool features** highlight the tools that are used to develop the DSL.
- **Process features** define how the DSL is used within projects.

The work of Lui et al (2010) highlights the lack of interoperable capabilities of DSL's along with the limited support from the available tools. In contrast to Langlois et al (2007) their approach to overcome these obstacles is to propose a service-oriented architecture approach. Their work proposes using WSDL to analyse web services to define the semantics of the DSL. Mernik et al (2005) also supports the concept that building a DSL is challenging and the tools for development are few. But from their work they do propose a design pattern that could ease the burden of developing a DSL. Again Barisic et al (2017) lament the lack of tools and techniques in designing a building DSL's. Their approach is to create their own DSL to specify the requirements of a DSL based system, their tool *USE-ME* supports the development of a DSL. In contrast the work by Vissel (2007) highlights a range of design patterns and guidelines to create reusable DSL templates.

The work of Thanhofer-Pilisch (2017) highlights the growing significance of DSL's in the ability to solve problems, but they do not look at how language DSL's have evolved. This importance of DSL in providing solutions has led to the development of a number tools to help create them. In 2016 Microsoft introduced a domain specific language toolset into their development tool visual studio. The open source project XText has been around since 2006 and is now part of the Eclipse Modelling Project. In 2009 JetBrains produced the first commercial IDE based on the MPS platform used to design DSL's. This product is an open source project under the Apache License So the tools exist to create the DSL and this project uses the XText framework to create the DSL.

Work by Kapferer (2019) shows how model transformation can be applied to DSL processing. The work also highlights that model-to-model transformation can also be applied to model to code as well as the reverse code to model transformations. This approach is also backed up by the work of Mens et al (2006). Work by the likes of Azuma et al (2003) and Goksu (2016) have highlighted the possible use of the Bloom taxonomy in the field of computer science. With Azuma et al (2003) the work not only shows how the Bloom can be applied to the field of computer science but also proposes a new taxonomy that could also be more applicable to software engineering problems. This chapter relies on the knowledge taxonomy proposed by Bloom to provide a strong foundation. In the case of Goksu the work concentrates on the use of the Bloom taxonomy in web-based expert systems to support learning within an educational group and proved to be more effective than just using traditional methods.

The work by Czarnecki et al (2003) highlights two alternative techniques to the embedding of a DSL in an independent application or language. The techniques described in the paper *staged interpreters* and *templates* according to Czarnecki et al (2003) can overcome the limitations of embedding. In their research the team looked at using MetOCaml, Template Haskell and C++. In their findings Czarnecki et al (2003) did express surprise at the number of DSL's that were implemented using the C++ programming language but also highlighted the flexible nature of the language and the subsequent flexibility that a language developed in C++ provided.

Zhao et al (2018) created DeepDSL for encoding deep learning networks and generating Java source code. At the core of DeepDSL are tensor functions and both tensor and scalar expressions. According to Hao et al (2018) there are a number of layers of processing to refine the Java code to ensure that it is optimised. The project highlights the processes and core components of the language as well as how it was created. This is in contrast to Czarnecki et al (2003) which just takes a high level view of the creation of a DSL.

Like Zhao et al (2018) the work carried out by Sujeeth et al (2013) provides a detailed view on the creation of the Forge DSL. Unlike DeepDSL, Forge is an embedded language based on the Scala programming language and uses the Delite framework.The work by Zhao et al (2013) provides some detail of the creation of the Forge DSL through the use of simplified code extracts. This chapter will follow the same path set down by both Zhao et al (2018) and Czarnecki et al (2003) by showing simplified code extracts as to how the language can be implemented.

Domain specific languages (DSL), as highlighted by the work of Klint et al (2009) and Starynkevitch (2011) can be used to simplify a specific task. In the case of Starynkevitch (2011), the work proposes using MELT a domain specific language in the GCC compiler to create plugins for the GCC Compiler. Starynkevitch (2011) argues that using the DSL reduces the complexity of creating plugins for the GCC compiler. Like the work proposed by Starynkevitch (2011), Klint et al (2009) uses RASCAL, a domain specific language to simplify the process of analysing source code. The review of DSL's carried out by Rodrigues et al (2017) also reiterates the idea that DSL's by stating that one of the aims of a DSL is to *ease the role of developers*.

The work by Klint et al (2009), Starynkevitch (2011), Starynkevitch (2011) and Rodrigues et al (2017) highlights the benefits of using DSL but their work is more generic than that required for this project. This project is looking solely at the use of a DSL for performing the translation from a natural language to a language capable of querying a database.

The Elasticsearch search engine has also been proposed as part of an interface between natural language and SQL. From the work carried out by Badhya et al (2019) they propose using the Elasticsearch as the training corpus for the conversion. The idea is that the keywords from natural language input statements are passed through Elasticsearch and compared to descriptive columns in the underlying database. The application uses an internal DSL called multi-match as part of the translation process. Again the work by Badhya et al (2019) provides a limited subset of the use of a DSL by this project. Badhya et al (2019) propose using an internal DSL to assist with the translation of natural language but the details on the implementation are not central to their proposal and as a result are scant on details.

When looking at the use of a DSL in the domain of a Natural Language Interface to Database (NLIDB) the work by Polosukhin et al (2018) introduces a DSL based on a Seq2Tree model. Their work also shows that this approach outperforms a sequence-to-sequence model in the conversion of a natural language into a language capable of searching a repository. However, the narrow approach taken by Polosukhin et al (2018) does not allow for the growth of the language by embracing other technologies or methodologies in the pursuit of converting natural language to SQL. The project also uses the DSL as an internal DSL which provides a solution as an alternative to a sequence-to-sequence model. The proposed use of the DSL within this project is to provide an interface between the end user and the underlying database. This interface provides a simpler interface with the aim of making data more accessible to users who now no longer need a high level of technical expertise to query and access data. More details of the DSL can be found in Chapter 5.

Few other projects in the field of NLIDB reference the use of a DSL. Most work consists of a solution based on either the use of augmented memory as in the work by McCann et al (2016), or that of Lin et al (2018) with their research into the use of semantic parsing. This work is in isolation essential to push the boundaries in improving the translation from natural language to a repository based query language. However, these approaches do not appear to provide an overall framework for building an inclusive approach, for this it appears that a domain specific language may provide a holistic framework.

4.3. Using Domain Specific Languages

Natural Language Interface to a Database solution has been around for a while, yet a review by Radu et al (2020) highlights only one project that uses a DSL and that was the SemQL from Guo et al 2019. Though work carried out by Skeggs and Lauria (2018) as well as Polosukhin et al (2018) have hinted at the use of a DSL as part of a solution to the problem of NLIDB.

Landauber et al (2016) proposed using an API as part of their solution. Their proposal uses a natural language command interpreter that models an API which acts as the interface for the translation. The SemQL domain specific language is a proposed DSL that is used as part of the decoder which was also part of the solution proposed by Skeggs and Lauria (2019). Yet the main focus on the Guo et al (2019) paper is the neural approach IRNet that sits behind the SemQL. The model proposed by Hanane et al (2020) appears to be a DSL light solution to converting the Arabic language into a language capable of querying a database. Though the hint of a DSL has been mentioned there appears to be no move to develop a language that can encompass the array of algorithms and approaches that have been developed over the years.

The review of domain specific languages carried out by Mengerink et al (2018) highlights how DSL's have been used to model systems. The conversion of natural language to a language which can query a repository is just another system. The paper by Namavari (2017) also highlights how a DSL can be used as part of a transformation process which is exactly what a NLIDB solution is, as it transforms one language into another. Namavari (2017) proposed using DAWPL (Digital Audio Workstation Programming Language) to generate, sequence and process sounds by using an API.

Domain specific languages are created to perform very specific tasks and as the work of Starynkevitch (2011) highlights they can be used to simplify what could otherwise be a difficult and time consuming process. DSL's are divided into two categories and are either internal or external, both variants of DSL have their own specific use. **Internal DSL** can also be referred to as an embedded DSL that is often embedded within another programming language or platform. The internal DSL LINQ as an example is used within the Microsoft .NET framework and the work by Boronat (2018) in creating YAMTL as a model transformation internal DSL of Xtend. **External DSL** in contrast are standalone languages that are implemented via an independent compiler or interpreter. HTML is a good example of an external DSL as its sole role is to render a web page.

The use of DSL's in the domain of natural language interface to a database refers often to the fact that SQL (Structured Query Language) is a domain specific language that is designed to manipulate a database. The use of a DSL in this scenario is that of the target as highlighted by Desai et al (2016). A target repository does not have to be a relational database but can also be a NoSQL or document based repository not supporting a standard RDBMS (Relational Database Management System) using SQL.

4.3.1. Internal Domain Specific Languages

As stated by Hinkel et al (2014) models can be transformed into other artefacts through code. Though according to Wimmer et al (2012) in their review of model-to-model transformations they highlighted that most transformations happen without the concept of a reuse mechanism making each transformation almost unique. Any similarity between transformations is there accidental rather than by design. Since this review was carried out work has been undertaken to simplify the process of model transformation and the use of DSL has been explored to allow for some reuse. Table 4.1 shows some of the most commonly used DSL’s.

Language	Usage
React	A language for building front ends to applications hosted by JavaScript
JQuery	JavaScript library designed to simplify HTML DOM tree traversal and manipulation of data.
Hyper Text Mark-Up Language (HTML)	Used within the internet to present page content through a browser.
Structured Query Language (SQL)	Used to query a database.

Table 4-.1: A list of commonly used internal Domain Specific Languages.

The work carried out by Boronat (2018) in developing the YAMTL DSL shows that it is possible for a DSL to be used as part of a transformation process as it is used to transform large models. This chapter is introducing the concept of a DSL as part of a transformation from natural language to a query language capable of querying a data repository. The work by Tisi et al (2018) also shows how a DSL can be used as part of a transformation process for large complex models, such as graph and rules-based models. The result of the work by Tisi et al (2018) was the introduction of the domain specific language CoqTL. The project highlights the transformative nature of CoqTL by transforming a Moore Machine into a Mealy Machine.

In addition the approach taken by Krikava (2015) is to propose an internal DSL as part of a solution to apply structural constraints to modelling languages. The team highlighted the difficulty of using general programming languages as well as using an external DSL. For this project their solution was to implement an internal DSL. Gulwani et al (2014) created an internal DSL that works within the bounds of Microsoft Excel. The aim of this project is to use natural language to help with the expressive algebra, map, reduce, join and formatting within a spreadsheet.

4.3.2. External Domain Specific Languages

External DSL's according to Riti (2018) are far more complex than internal DSL's to create. Part of the reason for this is that an interpreter or compiler needs to be created as part of the language structure that allows the DSL to run independently of any other structure. A number of external DSL's exist to perform a range of tasks, Table 4.2 lists some of the common external DSL's in common use with a brief description of the task each performs.

Language	Usage
HTML	A XML like language for displaying data within a web browser
SQL	A language to insert modify or extract data from a relational database
XML	A text human readable format for storing and transmitting data
NoSQL	An SQL like language for querying non relational databases

Table 4-.2: This above table lists some of the most common external DSL that are in use.

Both Sobernig (2020) and Storl et al (2015) have shown how Object Relational Mappers can be used to access NoSQL data stores just as well as dedicated Object-NoSQL Mappers. This work highlights the possibility of using object mappers to map data from a natural language onto the structure of potentially a relation and NoSQL database. Using the object mapper within a DSL could be the key to creating an interface between a natural language and a data repository.

4.3.3. Internal v External DSL's

Both internal and external DSL's have their place. According to Barringer et al (2011) DSL's can be difficult to extend which can prove limiting for a DSL as having initially created a DSL users will always want more features. The general argument for an internal DSL is to provide a quick easy interface making it easier to add functionality to the host application or language. The creation of the internal DSL is constrained by the host application. Internal DSL are more simplistic to write but knowledge of the host application can make access to the DSL difficult.

In contrast external DSL's can be harder to develop as they have no host and are required to work as a standalone application. The interface into the language can make external DSL's more user friendly. Cuadrado et al (2013) states that the DSL's have been created to improve efficiency in performing a specific task. The creation of Structured Query Language (SQL) is a prime example of this role in DSL's. SQL was designed to extract data from a database by providing an interface to the database which was easier to understand. DSL's in general are easier to learn than non-DSL's like C or Java.

4.3.4. Extensible Domain Specific languages

Barringer et al (2011) makes the argument that domain specific languages are difficult to extend but without the ability to change and adapt a DSL will have a limited life span. The aim is therefore to create a DSL that can change and adapt to users requirements over time. Since the Barringer et al (2011) paper a number of extensible DSL have been produced, a number of approaches have been used to achieve extensibility. Hinkel et al propose that a DSL should inherit tool support from the host language to allow for extensibility, though this is suitable for internal DSL's obviously for an external DSL this approach is not, therefore another approach needs to be used. According to Sutii et al (2018) designing a language to be modular is one of the key components of making the language extensible. The SteelCore extensible DSL came out of the work by Swamy et al and as Sutii et al (2018) proposed the use of a modular design. The PENROSE system from Ni et al (2017) uses yet another extensible DSL called Substance that renders mathematical notation. Heeren et al (2017) also created an extensible domain specific language, again by making it modular in design.

4.4. Designing the DSL

In this section we look at how to design the language structure grammar and syntax. Understanding design patterns in the creation of the DSL can focus the development process shortening the time and effort required in the creation of the DSL. This approach is backed up by the work carried out by Spinellis (2001). Their work identifies eight design patterns that should be used when creating a DSL.

- **Piggyback** looks for a host language or application to provide the foundation of the DSL around. The support from the host could potentially include expression handling and linguistic support.
- **Pipeline** describes how the DSL process marshals data through a process as described by Bentley (1986).
- **Lexical processing** looks at the language elements of the DSL which can be overcome by piggybacking on another language or application.
- **Language extension** shows how a DSL can be used as an extension of an exiting language introducing new features, components or core functionality.
- **Language specification** identifying which elements of the host language will not be implemented as part of the final solution.
- **Source to source transformation** looks at the compilation or interpretation of the language.
- **Data structure representation** looks at how data will be handled by the DSL.
- **System front end** looks at how the DSL can be integrated with other systems or programmed.

Most of the design patterns identified by Spinellis (2001) revolve around the grammar, syntax and general language structure of the DSL. These design patterns could be fulfilled by piggybacking off another language or application. The question of the language constructs would then be defined by the host language or application.

In contrast to Spinellis (2001), Zdun et al (2010) states that when implementing a reusable architectural design for a DSL states that there are three main decisions that need to be made for a DSL to be successful. According to Zdun et al (2010) these are, *development process*, *language syntax* of the DSL and whether the DSL is *internal or external*. For the purposes of this chapter the DSL is going to be external which just leaves the syntax and the development process which will be covered in this section.

4.4.1. Development Process

According to Zdun et al (2010) the DSL development process has three design patterns: Language Model Driven, Mockup Language Driven, piggybacking. For a language model driven design pattern the approach is to model the language features understanding the limitations of the language. With the Mockup Language Driven model the idea is to start with a concrete idea of the language structure and to then refine and build the model from that design. The piggybacking design model takes a host language or application as a starting point for the DSL. The idea is to identify existing elements within the host language or syntax and provide an interface to those features.

Each approach has upsides and liabilities which are set out in the Zdun et al (2010) paper. The approach taken by this chapter is the Mockup Language Design. The mock up for the language starts with understanding the Bloom taxonomy for Knowledge. Having an understanding of this taxonomy then enables the creation of a language mockup which is expanded in the next section.

4.4.2. Language Syntax of the DSL

The proposal for the language syntax is based on the concept of a question. After all, the purpose of a question is to elicit an answer. The language construct will follow the Bloom taxonomy for Knowledge as set down by Benjamin Bloom an American educational psychologist. The knowledge taxonomy is based around the standard questions **who**, **what** , **when**, **where**, **why** and **how** which are sometimes referred to as **wh** questions or the **5 W's (and 1 H)**. Bloom's taxonomy has been used in education for a number of years as a way of getting students to understand the subject matter, it has also been used by both journalists and law enforcement to try and understand the timelines of how and why events, stories or crimes unfolded.

The proposed language will concentrate on the first four questions: who, **what** , **when** and **where** . The remaining two questions **why** and **how** could be argued are too subjective. Further work would need to be undertaken to bring these two questions into the project.

There is a standard construct for a **who**, **what** , **when** and **where** based questions. In normal speech the **wh** word is the first article followed by an auxiliary verb (be, do or have), then the **subject** then finally the **main verb**. An alternative would be the **wh** word as the first article followed by model verb then subject and main verb. This chapter proposes to use these established language constructs as the foundation for the DSL, which establishes the ground rules for the language grammar.

This grammar can allow for the easy transformation from a knowledge based natural language question to a simple query capable of searching a repository. Using the grammar rules of the DSL the simple natural language query could be *what time is lunch served*. From this simple natural language question, Table 4.3 shows how the question is parsed into the salient elements of a knowledge based question.as well as the potential for how those mapping could be applied to a database.

Word	Word description	Database Mapping
what	the 'wh' article	N/A
time	an auxiliary noun	Database table
is	the auxiliary verb	N/A
Lunch	the subject of the question	Database table column
served	the main verb from the question	Database table column

Table 4-.3: The construct of a simple knowledge-based question.

The first column in Table 4.3 shows the words separated out. The second column shows the type of word (whether it is a noun or verb). The third column then maps the word to a construct in a database. The underlying table *meal_times* holds the times meals are served at a given restaurant is shown in Table 4.4. So by simply parsing the natural language question into the constituent parts and mapping those to a database construct such as a table or column it is possible to convert the input statement into the structured query *select time from meal_times where meal = 'Lunch'*.

Time	Meal
6:00	Breakfast
7:00	Breakfast
8:00	Breakfast
9:00	Breakfast
10:00	Mid Morning Coffee
11:00	Mid Morning Coffee
12:00	Lunch
13:00	Lunch
14:00	Lunch
15:00	Afternoon Tea
16:00	Afternoon Tea

Table 4-.4: A possible table structure that contains the meal times of a restaurant during the course of a day.

Not all knowledge based questions are so simple or can be converted to an SQL statement so easily. In the example above there would also need to be a mapping for the word or term *time* to the database table *meal_time*. Yet the algorithm that performs the conversion for a more complex conversion can be abstracted behind the use of a domain specific language.

4.5. The Language

Having chosen the approach that this chapter will follow in the creation of the DSL the next step is to select the toolset. A number of tools and frameworks exist to simplify the process of creating a DSL. Tools like the eclipse Xtext framework, Visual Studio from Microsoft MPS from JetBrains and the python based textX which is based on the Xtext framework. These tools simplify the process of creating a DSL by providing language parsing, linking, type checking and compiling covering most of the design patterns highlighted by Spinellis (2001). Bunder (2017) describes an approach to convert UML (Unified Modelling Language) models to a DSL using Xtext. Though the description in Bunder (2017) is brief the idea is to use the Xtext DSL tool in parallel with the Java Eclipse IDE to provide the heavy weight lifting.

Figure 4.1 provides a high level view of the process used to take the proposed DSL statement, validate the statement and tag the tokenised word elements of the statement with labels to identify how the words can be extracted into another language capable of querying a repository.

- The diagram shows that the parser uses the Xtext grammar file shown in code sample 4.1 to identify elements of the underlying database that have been identified as useful for the validation of the input statement.
- The input statement can then be validated to ensure that it meets the requirements of the prescribed language syntax.
- If successful the word elements or tokens from the input statement are tagged with standard OpenNLP tags to produce an output.
- If the input validation process is not successful the question becomes does the process raise an error. As it currently stands with this project the option is not to raise an error. As will be described chapter 6 and 7, the parsing statement can take a natural language statement and parse it into a language capable of querying a repository.

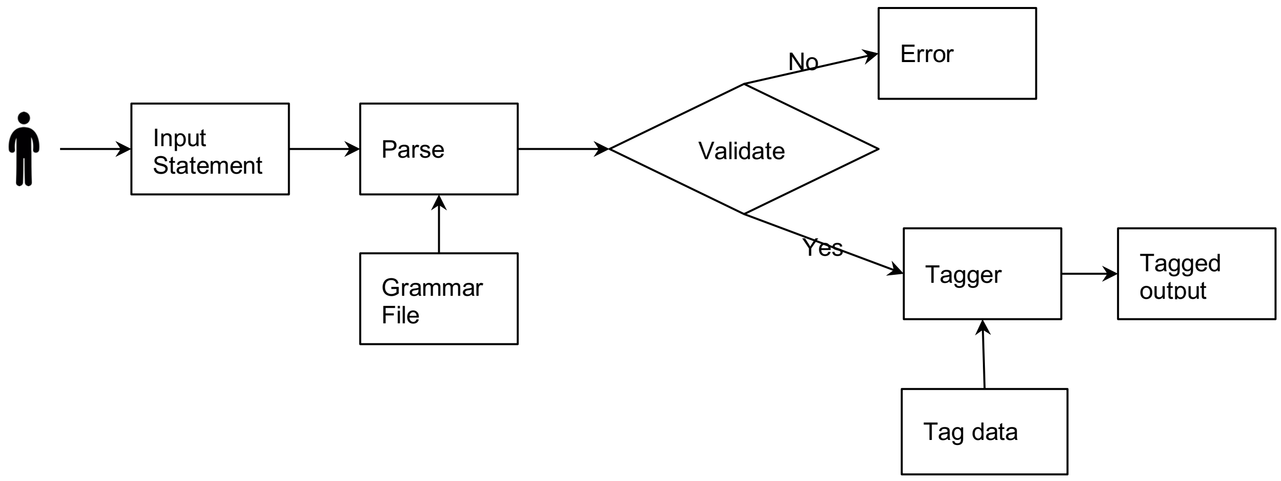


Figure 4-.1: Provides an overview of the processing required by the proposed DSL.

The concept behind creating the DSL is to provide some common structure to the natural language input statement. This structure can be used to reduce the complexity of the input statement making the parsing easier and quicker to perform. More details on the parsing of the input statement can be found in chapters 5 and 6.

4.5.1. Creating The Language

The language can be created using a tool such as Xtext and for the purposes of this chapter is to show how the language is prototyped using Xtext. Codes sample 4.1 shows an extract from an Xtext grammar file which is used to establish the feasibility of the language. From an example grammar file it is possible to see how a simple natural language query ‘*what time is lunch served’* can be parsed. The grammar file shows how the word tokens from the natural language question can be mapped to attributes within the underlying database, in an approach that is similar to a sequence-to-sequence model.

```
Model:
    declarations += Declaration*;
Declaration:
    Rule;
Table:
    name=ID;
Meal:
    name=ID;
Rule:
   ‘what’ description=STRING'

   ‘time’ time=[Table|QualifiedName] 'is'

   ‘lunch’ lunch=[Meal|QualifiedName];

QualifiedName:

    ID('.'ID)*
```

Code Sample 4-.1: A simplified grammar file for the DSL showing how the file can be constructed to handle a natural language statement.

The grammar file has to know and understand the structure of the database it is expected to query. Therefore the grammar file needs to be created from the underlying database. The structure of the grammar file contains the elements used to extract the salient points from the underlying database.

The first rule in this grammar file is a *‘Model’*. This model is made up of a single or multiple *‘Declarations’*. In the case of the grammar file in code sample 4.1 the model contains a single declaration which is a *‘Rule’*. The Rule has three elements the first is a string which from the example is the string *‘what’.* In a more complete grammar file this should include the other ‘wh’ words as defined in Bloom's knowledge taxonomy (see section 4.4). The second rule seen in line 11 is the word *‘time’* which should also be present in the input string , and should be qualified by the trailing word *‘is’* . The word *‘time’* has also been identified as having a rule named *‘Table’* associated with it. The text *‘lunch’* also has a rule associated with it, called *‘Meal’*. The rule is shown in line 5 and 6. attribute of the underlying database. The grammar file should contain elements of all the keywords that should be in the corpus of questions that can be asked of the database.

The more complex the grammar rules used to define the DSL the more complex the grammar file. For the simple example that is highlights in codes sample 4.1 the content of the grammar file can become unwieldy and prone to errors.

The idea behind the language is to provide a framework on which algorithms from other research projects can be implemented. This section looks at how algorithms can be implemented within the DSL. The algorithms to be implemented will be for the parsing of the natural language statement, the thesis will then look at how to implement a sequence-to-sequence model.

4.6. Parsing the Natural Language Statement

There are two stages of parsing required for creation of the DSL, the first is to ensure that the input natural language statement meets the language constructs or grammar defined within the language specification. The second parse required takes the elements of the natural language input statement and identifies the database elements that are associated with those input elements. This becomes the foundation for the creation of the resulting SQL statement. This section of the thesis splits out the two parses and shows how they are to be treated.

4.6.1. Initial Parse

The first step when creating the domain specific language is to parse the incoming statement to ensure that the incoming statement meets the requirements of the DSL. This would include the use of reserved words, the grammar involved with the DSL and any potential mark to identify the extent of the input statement along with any special characters to define comments. Chapter 6 details how the design of the DSL can simplify the process of parsing the input statement, for a simplified version looking at *who*, *what*, *where* and *when* questions the standard grammar is the *wh* word as the first article followed by an auxiliary verb (be, do or have), then the subject then finally the main verb. An alternative could be the *wh* word as the first article followed by model verb then subject and main verb.

With these simplified ground rules in place we can start to build the parser. Using the Stanford CoreNLP parser to parse an incoming request using the simplified natural language questions *What time is lunch served?* and *When is lunch served?* The output from the tagger is shown in Table 4.5 and the definition of the assigned tags are shown in Table 4.6.

Word Token	**Part of Speech Tag**
What	WDT
Time	NN
When	WDT
Is	VBZ
Lunch	NN
Served	VBZ

Table 4-.5: Shows the output from parsing both the simple questions. The output contains the word tokens and the part of speech tags associated with each word.

Part of Speech Tag	**Tag Definition**
WDT	*Wh* determiner
NN	Noun
VBZ	Verb, 3rd person singular present
NN	Noun, singular or mass
VBN	Verb, past participle
VBZ	Verb, 3rd person singular present
WRB	*Wh* adverb

Table 4-.6: Shows the definition for the associated tags assigned to the natural language statement.

Both questions are valid in English and both are valid in the context of the context of the DSL. Both statements start with the *wh* article. In the first statement the next word is *time* which can be either a verb or a noun. The tagger has incorrectly defined the word as a noun, but it still meets the requirements of the language. This is then followed by a verb *is* then the noun *lunch* which is the subject lastly by the main verb *served*. Refining the parsing of the incoming natural language statement to ensure that the word *time* can be parsed correctly as a verb instead of a noun is a requirement.

4.6.2. Secondary Parse

Having identified that the input natural language statement is valid the next step is to reparse the statement ensuring that the database elements can be identified from the input statement which can then be used to create the resulting structured query language statement.

This section will show how to implement the shallow parsing model created by Skeggs and Lauria (2019) which is also Chapter 6 of this project. The approach proposed was to take a shallow parsing approach to querying the natural language statement. The idea behind shallow parsing as described by Li and Roth (2001) is to chunk a sentence into base components whether that is phrases or words. The work by Li and Roth (2001) goes on to show the effectiveness of shallow parsing as an approach for extracting meaning from a natural language statement.

The proposal is a four step process, from parsing the incoming natural language statement to creation of the structured query language statement.

- The first step is the tagging of the input statement with the appropriate tag. For example the statement *What time is lunch served?* the tagging would look like *What_IRR time_AP is_IRR lunch_NP served_IRR ?_IRR .* The tagging process uses a slightly modified part of speech tagger from the OpenNLP project. The standard OpenNLP tags are used to identify which elements in the incoming statement are identifiable. There is also an additional custom tag *'_IRR'* which is used to identify which word tokens in the incoming natural language statement can be ignored by the conversion process.

- The next step would be to take the identified key words *time* and *lunch* and try to map them to elements within the database to identify the select part of the SQL query and come with the result *select * from time*.

- The next step is to try and create the conditional part of the SQL statement with the result *where meal = lunch*.

- The final step is to create the full statement *select * from time where meal = lunch*.

- The functions used to create the SQL statement are hidden within the domain specific language but it allows for the shallow parsing approach with keyword identification to be implemented.

4.7. Implementing the DSL

The next step is to show how the DSL is to be implemented using the parser defined in the previous section. The advantage of using the DSL is that the structure of the incoming natural language statement is set to a predefined language construct. This language construct reduces the complexity of parsing and processing free-formed natural language text.

4.7.1. Validating the DSL Statement

Having defined that the DSL is conforming to Bloom's taxonomy of knowledge parsing the incoming natural language question to ensure the correct language structure is met as set out in Section 4.4. The code to parse and validate the incoming natural language statement is shown in the code extract below.

```
from nltk import pos_tag

from nltk import RegexpParser

input_text = ‘What time is lunch served’

print(pos_tag(input_text))
```

Code Sample 4-.2: The code extract shows the first pass at parsing the incoming natural language statement. It is this piece of code that determines whether the input statement is in the correct format.

The output from the code above is

```
[(‘What’, ‘WDT’), (‘time’, ‘NN’), (‘is’, ‘VBZ’), (‘lunch’, ‘NN’), (‘served’, ‘VBN’)]
```

A simple pattern match to the target structure will determine that the natural language statement meets the desired input. Having validated that the incoming natural language statement is in the correct format the next step is to parse the incoming statement into a language capable of querying a data repository. The details of how this chapter proposes to parse the incoming natural language statement can be found in chapters 5 & 6.

The concept of the DSL is to simplify the translation process from natural language to a language capable of querying a repository. Structuring the input language into a predictable format reduces the complexity of parsing the natural language input statement.The question then becomes what happens when the input statement does not match the expected format. The two possible approaches are:

1. Ignore the error and parse the statement as if the input statement does match the expected input.

2. Display an error to the user stating that the format is not of the expected type.

The approach being taken by this project is to ignore the error and continue parsing. Even though the DSL is part of the project the language is currently designed not to be strongly typed and the format as well as the structure of the language are not strongly enforced. The idea being that the language syntax should not become onerous to learn enabling the democratisation of the language. As can be seen in Chapter 5 and 6 is that the parsing algorithm that has been developed as part of this project can parse natural language statements.

4.8. Conclusion

The idea within this chapter was to initially create a simple prototype of the DSL by using a simplistic parsing approach with a sequence-to-sequence model to map the word tokens from a natural language input statement onto the comparable database assets. This simplistic sequence to sequence model has limitations as work by the likes of Guo and Gao (2018) and Su et al (2018) has highlighted. A solution based on parsing a natural language text would need to be incorporated into the model to fulfil the purpose of the DSL being extensible.

This chapter uses Xtext to quickly create and prototype a functional DSL. Advantages of using a tool like Xtext is that the tool creates and manages an environment in which the DSL can be developed and accessed. The downside to this approach is that the grammar file and parsing tools within Xtext are limited. It soon became apparent that these limitations would not allow the flexibility required to create an extensible DSL. Therefore a new approach is required which requires that the parser, grammar and development environment would need to be created explicitly for the new DSL.

The section 4.5 highlighted the limitation of using a tool like XText to create the DSL. The complexity of the Xtext grammar can make creating and managing the file when the grammar rules become more complex extremely difficult. This increased complexity inherent in the Xtext grammar file goes against one of the underlying principles of this project in that the DSL should be extendable. The project has already shown how the natural language statement can be parsed into a language that can be used to extract data from a repository. The next two chapters show how the incoming natural language statement can be parsed meaning extracted and ultimately converted into a language capable of querying a data repository. This leaves the Xtext component just the task of validating the incoming natural language statement to ensure that it is compliant with the desired DSL language structure.

The simplicity of the Bloom Knowledge Taxonomy on which the DSL is based makes parsing the incoming natural language statement simple. The objective of this parse is primarily to ensure that the input statement matches the approved grammar. The more detailed parse for converting the natural language statement into a language capable of querying a database is covered in both Chapter 5 and Chapter 6.

Having seen the uses of Domain Specific Languages and how they can be used within a transformation, the next step is to create a DSL that can be used not just as a language to convert natural language to SQL but also as a wrapper to host the algorithms currently in research projects. The work by Gulwani et al (2014) comes close to the aim of this project by showing how an internal domain specific language NLyze, can be used to manipulate a spreadsheet. But this project has failed to show how this approach can be used to extract data within a large data repository and is also tightly bound to Microsoft Excel which makes the solution non interoperable. With work such as the AskMe NLIDB system from Llopis and Ferrandez (2016) which provided a framework for the other research projects the team did not create a domain specific language but more of an application.

The work of Gulwani et al (2014) and Badhya et al (2019) show that domain specific languages have a place in the translation of a natural language to a language capable of querying a database. But their work also highlights the fact that to fully create an extensible DSL capable of working with a range of repositories is to create an external DSL. Both solutions proposed by Gulwani et al (2014) and Badhya et al (2019) are internal DSL's which bind them inextricably to their host, therefore, to make an interoperable DSL it has to be external.

This project looks at providing a domain specific language as an interface to a natural language interface to a database. From the literature review carried out as part of this project there appears to be no mention that this approach has been taken by another project. Most of the projects reviewed for this thesis have provided no details as to how the proposed algorithm could be implemented. This project is proposing developing a DSL that can act as the interface between the user and the underlying data. Using the DSL streamlines the translation process as the inbound natural language text comes into the system in an understood and predefined manner thereby streamlining the translation process. Using the DSL also has the capability to reduce the complexity of setting up the system as parsing the incoming statement will be in a predefined structure with a high degree of known words. The following chapters highlight how the data parsing is part of the translation process.

5. A Shallow Parsing Approach to Natural Language Queries of a Database

5.1. Introduction

Two of the key goals of this thesis are to improve the speed of converting natural language into a language capable of querying a repository, and to also improve the returned search results. This chapter looks at the use of shallow parsing or as it is sometimes referred to as part of speech as an approach to solving these issues. The concept behind shallow parsing is to extract the important key words from the incoming natural language query and map those keywords to attributes within the underlying database. This approach ignores the natural language concept of nuance.

The performance and reliability of converting natural language into structured query language can be problematic in handling nuances that are prevalent in natural language. Relational databases are not designed to understand language nuance. The natural language query *'who are my 10 worst customers'* can cause problems with translation to an SQL query and what does the word worst mean. Does it mean the customers who buy the least amount of goods from me or the customers who return the highest percentage of the goods bought, or the ones whose products cost more to produce. It could even mean something completely different. The structure of a SQL query when retrieving data is to select a value from a column within a database table. There is no room for language nuance in the query '*select name from customer order by qty asc limit 10'*. This query will order my customers by the quantity of goods sold. It will not necessarily highlight my worst customers as new customers are likely to have bought less goods.

Therefore the question is must we try and handle language nuance when converting from natural language to a language that can be used to query a database. Within this thesis what is being proposed is that language nuance should not be considered when converting natural language to a language capable of querying a database. There is no language nuance within the target language of SQL and the risk of misinterpreting language nuance is too high. The idea is to therefore create a DSL which can reduce the need and reliance on language nuance. By formally structuring the format of the language and grammar of the incoming statement it can become easier to perform the parsing of the input statement and convert the statement more accurately to a language that is capable of querying a data repository.

This chapter is based on the Skeggs and Lauria (2019) paper published in the Journal of Software Engineering and Applications. The Skeggs, Lauria (2019) paper proposes an alternative solution to that proposed by the likes of Bais et al (2018) and Jwalapuram & Mamidi (2017) for the conversion of a Natural Language Query into a Structured Query Language (SQL) capable of being used to search a relational database. The proposed process uses the natural language concept, Part of Speech, to extract keywords from the input natural language statement that can be used to identify database tables and table columns. Taking this approach removes the ability of the system to handle language nuance but does identify the key elements of the natural language statement that can be used in the conversion to an SQL statement. As an example of this the query *'which customers have bought the most goods this month'*, using Part of Speech the sales table is required for the search and the columns used within the query would be the quantity customer number and data fields.

The solution being proposed in this chapter uses the Apache OpenNLP application to enable the NLP parsing. The OpenNLP standard configuration is enhanced with additional configuration files to assist in the translation from natural language to query language. Having used part of speech processing within OpenNLP to identify which tables and which columns contain the pertinent data the next step is to create the SQL statement. A more detailed description of the architecture being proposed by this project is discussed in section 5.3.

5.2. Related Work

With the quantity of real-time data and the speed of data increases the need to search and extract data from multiple sources is becoming more important. Natural Language Processing can be useful for converting natural language text into a formal structure that can be processed by a computer program.

The growth in size and importance of data within society has led to the development of a new range of tools to query, examine and analyse data. Even the increasing use of tools like Siri, Bixby, Alexa and Google Assistant to perform searches is changing the way users look for information. Amazon is one of the largest retailers, AirBnB is one of the largest hotel groups but neither company owns a single store or hotel. Both organisations class themselves as data companies. The rise of the importance of data is driving a need for new tools and techniques for managing the storage and retrieval of information from data repositories. With large quantities of data stored within databases or database backed repositories providing an interface between a non-technical user and data is becoming increasingly important.

The use of a natural language interface to a database enables non-technical users to search a database using natural language statements, whether that is the spoken or written word. The Natural Language Interface to Database (NLIDB) provides the interface between a natural query and a structured data query language like SQL. This allows for data retrieval without the need for technical knowledge or a detailed understanding of the Structured Query Language (SQL) or even knowledge of the underlying database.

A number of systems described by Reshma and Remya (2017) such as LADDER, CHAT-80, NaLIX and WASP have all been developed to become the interface between natural language and the database but none of them have come into mainstream use. The issues these tools have struggled with revolve around natural language complexity. The most common one of these complexities has been understanding the language nuance of the natural language statement as described by Bais et al (2018), Florin et al (2017) and Deuter (2015). Other issues have revolved around the performance of the interface in converting the natural language query not only in a timely fashion but also with the accuracy of the returned results which was initially highlighted by Gallant (1990) but has also been raised by Joshi and Akerkar (2008).

This chapter is proposing a solution to solve both the language nuance highlighted by Florin et al (2017) and Kiev et at (2011) as well as the performance issues with the use of shallow parsing as discussed by Joshi and Akerkar (2008). The use of shallow parsing, also referred to as part of speech, negates the requirement for an understanding of language nuances, as key words are extracted from the input statement and used within the conversion process. The shallow parsing approach being proposed by this chapter is the use of keywords. This approach first proposed by the Ratnaparkhi (1996) is used to identify characteristics of the input statement that are important for the search. In contrast to Ratnaparki's approach this project not only identifies the keywords that would be useful in the translation process, but maps the keywords to tables and columns within the tables. This chapter will introduce the use of an index file containing keywords extracted from the underlying database that identify the tables and associated columns. It is this approach that helps build the performance in translation from natural language to SQL. This builds on the work of Jwalapuram & Mamidi (2017) who are among a number of authors who have carried out research into using keywords to enable NLIDB based systems to perform searches. Unlike that used by Jwalapuram and Mamidi (2017) this project uses Part of Speech (POS) processing in conjunction with an index file which allows for individual words to be extracted from the natural language query. The individually extracted words can then be used to create the query for the NLIDB solution. Details of the architecture for the proposed solution can be found in section 5.4.

5.3. Football Events Data

To test the performance of the NLIDB application an open data set was selected for testing and benchmarking. The website Kaggle.com has several openly available large datasets that can be used freely. The Football Events dataset was chosen and is available via the following link (https://www.kaggle.com/secareanualin/football-events). This dataset was chosen as it contains two tables which ensure that the feature to join the two tables together can also be tested. The concept of being able to join two or more tables together is important as this feature is often useful when searching data repositories as data can be held across multiple tables.

The dataset comes in the form of two comma separated value (CSV) files which are labelled EVENTS and GINF. The events recorded in the tables cover 9074 football games from across Europe. The two tables are in CSV format which makes it easier to load into a database whether that is a no-SQL or RDBMS version. The two tables within the data set are:

- The EVENTS table as shown in Table 5.1 contains details about each game. The data has been scrapped from bbc.com, espn.com and onefootball.com and has 941009 recorded items.

- The GINF table, details are shown in Table 5.2 contains metadata and market betting odds for each game and contains 10112 entries. The odds for the dataset were supplied by oddsportal.com.

The two tables can be joined using the common key ID_ODSP, which is the unique identifier for the game.

Column Name	Description
ID_ODSP	Unique id of the game
ID_EVENT	Unique identifier of event (ID_ODSP + SORT_ORDER)
SORT_ORDER	Chronological sequence of events in a game
Time	Minutes into the match
Text	Description of event
EVENT_TYPE	Primary event. 11 unique events (1-attempt (shot), 2-corner, 3-foul, 4-yellow card, 5second yellow card, 6-(straight) red card, substitution, 8-free kick won, 9-offside, 10-hand ball, 11-penalt conceded)
EVENT_TYPE_2	Secondary event. 4 unique events (12-key Pass, 13-failed through ball, 14-sending off, 15-own goal)
Side	Home or away team (1-home, 2-away)
EVENT_TEAM	Team that produced the event (In case of Own goals, event te is the team that beneficiated from the own goal)
Opponent	Opposing team
Player	Player involved
Player 2	Player involved
PLAYER_IN	Player that came in (only applies to substitutions)
PLAYER_OUT	Player substituted (only applies to substitutions)
SHOT_PLACE	Placement of the shot (13 possible placement locations, available in the dictionary, only applies to shots)
SHOT_OUTCOME	4 possible outcomes (1-on target, 2-off target, 3-blocked, 4-hit the post)
IS_GOAL	binary variable if the shot resulted in a goal (own goals include
Location	Location on the pitch where the event happened (19 possible locations, available in the dictionary)
Body Part	Body part ball touches (1-right foot, 2-left foot, 3-head)
ASSIST_METHOD	In case of an assisted shot, 5 possible assist methods (details the dictionary)
Situation	In case of an assisted shot, 5 possible assist methods (details the dictionary)
FAST_BREAK	Did a fast break occur

Table 5-.1: The EVENTS table describes the structure of the events database. This table is joined to table 5.2 on the unique identifier for the game, ID_ODSP

Column Name	Data Type	Description
ID_ODSP	String	Unique ID of the game
LINK_ODSP	String	Link to odd sportal page
ADV_STATS	Boolean	Availability of advanced statistics
Date	Date	Date of event
League	String	The league the match was played
Season	Number	The year the season finished
Country	Number	The country the match was played in
Ht	String	Home team
At	String	Away team
Fthg	Number	Full time home goals
Ftag	Number	Full time away goals
ODD_H	Number	Highest home wim market odds
ODD_A	Number	Highest away market odds
ODD_OVER	String	Highest over 2.5 market odds
ODD_UNDER	String	Highest under 2.5 market odds
ODD_BTS	String	Highest both teams to score market odds
ODD_BTS_N	String	Highest both teams not to score market odds

Table 5-.2: The GINF table describes the features of the GINF table. This table is joined to table 4.1 on the unique identifier for the game, ID_ODSP

5.4. Proposed Configuration

This chapter is proposing to use three index files to aid the conversion from natural language query to SQL. The files being proposed are the Grammar file, Join file and Index file. The use of these files ultimately describes the structure of the underlying database which will become the target for searching, while providing an index-like data structure that can be used to identify the database table(s) and table columns relevant for the database search.

The files described in this section can be created either manually or through scripting. The grammar file should be created through the collection of queries that have been used to query the underlying database. With a historic record of prior questions, the grammar file can be enhanced.

Figure 5.1 shows an overview of the proposed architecture for the NLIDB solution being discussed in this chapter. The details of which will be expanded in this section but the steps are highlighted:

- Parse the input statement into tokens: The natural language query is broken into word tokens using the appropriate tagger. This chapter proposes the use of two taggers OpenNLP Part of Speech tagger and the OpenNLP Names Tagger. Both taggers can be used in parallel each to perform the task of identifying key words that are useful to the conversion process and to identify names that are within the input statement.

- Parse Tokens and tag. The next step is to take the word tokens which is the output from the previous step and apply a tag. The grammar file contains the details of the tags to be applied to each token.

- Remove stop words. The next step is to remove the stop words from the process as these words add nothing to the conversion process.

- Parse Process. The parse process uses a template for the standard SQL statement and creates the final SQL statement using the join file for searching multiple tables.

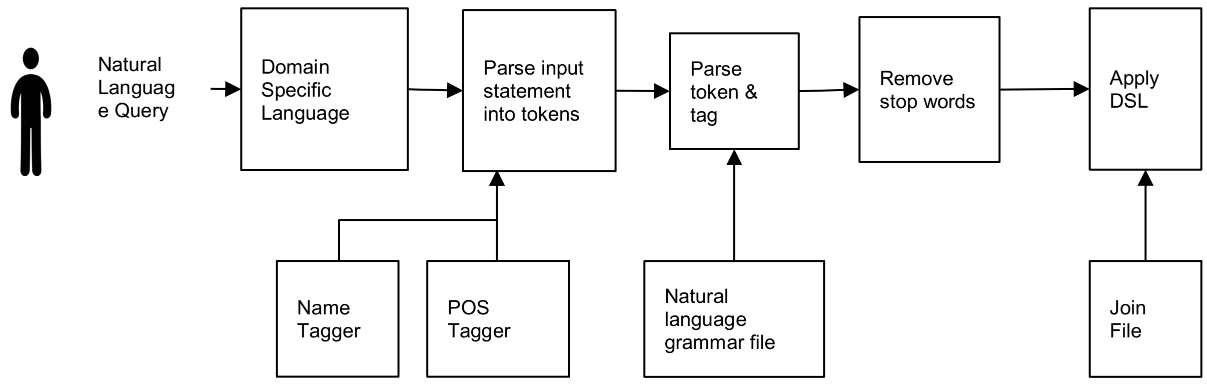


Figure 5-.1: Shows an overview of the proposed system. The processes that will be applied to the natural language statement as it is converted into a language capable of querying a repository. Here the taggers are separated into name and part of speech.

This section will look in depth at the process used to convert the natural language statement into a language capable of querying a data repository. As part of that process the configuration of the configuration files will also be explained.

5.4.1. Parse Input Statement

The simplistic approach to parsing the natural language statement into tokens is to remove punctuation from the statement and split each word into an array of tokens. A simple approach is shown in Code Sample 5.1:

```
#Simply split the string on the space character
def tokenizer(query):
    tmp= re.sub('[^A-Za-z0-9 ]+', '', query)
    return tmp.split()
```

Code Sample 5-.1: A sample Python script that can be used to parse an incoming natural language statement into tokens. In effect the script splits the incoming string into an array of words.

5.4.2. Parse Tokens and tag

The database extraction process which provides data for the three configuration files manually extracts data from the target database. Though the process is manual there is nothing about the structure of the configuration files nor the data used by the files which stop their creation from being automatic. The process was completed manually as the dataset was small enough for this task to be completed.

The first of these is the Apache OpenNLP grammar file which is used to identify words in the natural language query. The content from the database is used to create the grammar file, column names from the database tables are tagged with N and the database tables are tagged with AP within the grammar file, an example of a grammar file can be seen in Figure 5.2. Separate tags are assigned to each word which identifies words of importance that can be labelled as either a table name or column name. The convention for tags is that VB identifies a verb, N for noun and ADJ for adjective, a full list of tags can be found in Appendix A. The list of tags is used by convention rather than being statically defined, therefore custom tags can be created to fulfil a specific task. This chapter uses a custom tag IRR to identify words that are irrelevant in the conversion from natural language to query language. In the example used for this chapter, the grammar file is constructed from entries from both the GINF and EVENTS tables. Questions posed to the application are also used as part of the grammar file. Table 5.3 lists the column names from both source files that are used within the grammar file. Sample code that can be used to achieve the tagging required by this process is shown in **Code Sample 5.2**

```
def tagger (tokens, patterns, names):
    name_tagger = nltk.RegexpTagger(names)
    regexp_tagger = nltk.RegexpTagger(patterns, backoff=name_tagger)

    tagged=regexp_tagger.tag(tokens)

    for x in range(len(tagged)):
        if (tagged[x][1] == 'out' and tagged[x+1][1] == 'in'):
            tagged.append((tagged[x][0] + " " + tagged[x+1][0], 'PC'))

    return tagged
```

Code Sample 5-.2: Shows how the tags can be applied to two taggers. In this example a Name Tagger is being applied as well as the tagger library which is highlighted in Figure 5.2.

The index data extracted from the GINF table contain 10,643 entries which are made up of the original entries with some additional data. Entries from the Events table create an index file with 1201 unique entries in the data. The structure of the table is made up of potential questions that could be posed to the NLIDB application. Each word is assigned a tag representing how that word should be treated. The tags follow the appropriate word and are separated from it by an underscore.

The grammar file (an extract of which is Figure 5.2) for this chapter uses a couple of tags, IRR which stands for irrelevant and ensures that the word will be ignored in the conversion from natural language to structure query language. The IRR tag is defined as being words or values not found within the underlying database as either table names, columns or values.

NP, which signifies that the word is important in the conversion process and states that is a value of significance and will be used within the search as this is the search criteria. Words tagged with AP signify the table that must be searched.

Events	GINF
ID_ODSP	ID_ODSP
Side	Date
EVENT_TEAM	League
Opponent	Season
Player	Country
Player 2	Ht
SHOT_PLACE	At
SHOT_OUTCOME	Fthg,±Ftag, ODD_H, ODD_D

Table 5-.3: Lists the entries extracted from the database for inclusion into the index file. The table also highlights the structure of both data tables. It can be seen that the column ID_ODSP is common between both tables and can be used to join them.

```
Which_IRR event_AP has_IRR
Which_IRR opponent_AP has_IRR ustaritz_NP faced_IRR
What_IRR are_IRR the_IRR odds_N on_IRR a_IRR game_IRR with_IRR an_IRR event_AP
involving_IRR caro_NP
What_IRR are_IRR the_IRR odds_N on_IRR an_IRR event_AP that_IRR caro_NP is_IRR
involved_IRR with_IRR
```

Figure 5-.2: This shows an extract from the grammar file showing the data structure. Finally, the tag N defines which column could potentially be used to extract data.

The grammar file highlighted in figure 5.2 was created manually having been built up from a list of historically asked questions and the content of the underlying database. The tagging used replicates the process used by the OpenNLP tagger. The figure shows that each word in the natural language statement has an associated tag. The tag *'_IRR'* indicates that the word has no associated tag whereas the word *odds* has been identified as being a noun by the tagger if it has the tag *'_N'.* The full list of tags can be found in Appendix A. The idea behind creating the file manually was so the tags could be tested to ensure that the tagging process could be optimised and tested.

t_IRR are_IRR the_IRR odds_N on_IRR a_IRR game_IRR with_IRR an_IRR event_AP

In this project the grammar file is currently created manually but there is nothing within the file that prevents its creation through automated scripts. The reasoning behind creating this file manually was to allow for testing and refining of the grammar file to optimise the conversion process. The file contains elements from the database being searched; an extract from the index file is shown in Figure 5.3. The data is made up of three columns; the first column shows the relationship between the table, the table column and the database value. The index file uses the same tags as the grammar file to identify elements that are within the database such as the tables, columns and values. Figure 5.3 shows that the AP tag is assigned to the value event, this represents the table. The second value is *player* which is assigned the tag N, which represents the column in the table. The third column shows a value in this case the name of a player (Abdoulaye Diaby) which has been assigned the tag NP.

From this, information the query is beginning to be built and simplistically the query is *"select * from event"*. The second column describes which variable from the table to use as part of the condition. In the example below, the word *player* is identified as a noun which can in this example identify the columns from the table *event*.

```
Event_AP player_N Abdoulaye#Diaby_NP
Event_AP player_N Abdoulaye#Faye_NP
Event_AP player_N Aboubkra#Kamara_NP
Event_AP player_N Adam#Federici_NP
Event_AP player_N Alberto#Garcia_NP
Event_AP player_N Aleksandr#Iakovenka_AP
Event_AP player_N Alemde_NP
```

Figure 5-.3: Extract from the grammar file.

This now means that the query is "*select * from event where player* =". The only element missing is the value to search on or in this case the player's name. This information comes from the third column labelled NP. From the extract in Figure 5.3, there is an extract of abdoulaye#diaby_NP, so the final query is now "*select * from event where player = 'abdoulaye diaby*"'. The use of the # symbol between the first and last name of the player makes it easier for this simple application to identify names. It is also possible to use a name tagger to identify the names of the players.

5.4.3. Join File

The above example shows the first step into parsing a natural language query into a simple SQL statement. Not all queries are that simplistic as some will require that tables are joined to extract the required data. A key aspect is how the joins between tables can be identified not just from the natural language query but also from the table structure. One possible solution is from the configuration within the grammar files.

This chapter suggests using a join file which lists the table and the primary key for the table. This table (see Figure 5.4) allows two tables to be joined. The table contains two entries which are the table name and the primary key of the table. In the example below, both the Event table and the GINF table can be joined and both share the same primary key (ID_ODSP).

```
# load the join file into a dictionary
join = {}
with open("file.txt") as f:
    for line in f:
        (key, val) = line.split()
        d[int(key)] = val

# extract the join key for a given table
join[table_name]

#extract the table from a given join key
for key, value in join.items():
     if value == join_key:
             return(key)
```

Code Sample 5-.3: This python code shows that the table name or the join key can be used to extract the attributes used in joining multiple tables together.

The join details from code sample 5.3 are loaded into a python dictionary called *join*. The look up becomes a simple function join[table_name] to return the join key. From a *join* key the look up becomes a simple loop. The process for creating the join file is manual but as discussed above in the section titled Proposed Configuration there is the possibility of automating this process. The caveat when creating an automatic script is to identify which tables have an identifiable relationship as well as what contrives to make that relationship. In the simple case discussed within this chapter, the relationship is easy to identify and easy to create as only two tables exist. In larger more complicated database environments identifying these relationships may be harder to identify. Using deep learning techniques to identify which tables are related and how that relationship exists may be required for an automated script.

5.5. Parse Process Conversion Steps

Having highlighted the components of the conversion process, the next step is to show how the whole process works. The solution proposed by this chapter allows for the example natural language query "*What are the odds on a game involving Caro?*" to be converted into an SQL statement. The starting point for the conversion process is a simple SQL template that defines the basis of a select query. The template is:

SELECT <parameters> FROM <table name> JOIN <table name> ON <field name> = <value> WHERE <field name> = <value> AND <field name = value>

Using the following steps, the parse process takes the SQL template, extracts relevant data from the natural language statement and transposes values where appropriate onto the template. The steps to perform this process are highlighted below.

```
event=ID_ODSP
ginf=ID_ODSP
```

Figure 5-.4: The join properties file lists the table name with the primary key which allows multiple tables to be joined.

- Tag the natural language statement. The OpenNLP tagger process takes the original statement and labels each word component with a natural language tag. An example output from the tagging process will look like.

  what_IRR are_IRR the_IRR odds_NP on_IRR a_IRR game_IRR event_AP involving_IRR caro_NP.

  The code used to produce the output would use NLTK library (such as the Python Natural Language Toolkit or the OpenNLP library).

```
import nlp_library
query_text = '<somestring>'
nlp = new nlp_library
processed_string = nlp.word_tokenizer(query_text)
print(processed_string)
```

- Looking at Figure 5.1 the grammar file identifies that the word event has the tag "AP". The conversion process identifies AP as a table. Using this information, the first part of the query is "*select * from event*".

- The next step taken by the proposed system is to identify that the query should join the *events* and the *GINF* table together as the query is asking for odds from the GINF table and player (Caro) from the events table. The join table specifies that the tables' event and ginf are joined by the column ID_ODSP. This creates the where clause "*where event.id_odsp = ginf.id_odsp*".

- The final step is to identify that the player being searched for is "caro" (see above). This gives the final part of the query *where player = "caro''*.

- The query can now be joined into *select * from events where event.id_odsp= ginf.id_odsp and player = "Caro"*.

- Currently, the select statement just uses "*select * from*". The next step is to retrieve just the requested data or columns from the database. Through the use and application of machine learning techniques it is anticipated that select everything could be reduced to selecting only relevant columns from the query. The following statement shows the structure of the target SQL statement where *P* is the parameter to be '*select from the target table T select <P_1>,<P_2>, <P_n> from <T>'.*

5.6. Training the Model

Having created the model the next step is training the model. The OpenNLP toolkit model uses machine learning algorithms at its core. Having created the configuration files to be used as a model, the next step is training the Apache OpenNLP model. Training the model is an important aspect of the Apache OpenNLP process. The mathematical models used by the OpenNLP application require that the model is trained. As the model being used by this project is a bespoke model, training allows the model to perform the word tagging using the grammar file more accurately than would have been otherwise achieved. The machine learning models used by OpenNLP for training include maximum entropy and perceptron-based machine learning.

The use of a maximum entropy model as described by Ratnaparkhi (1996), ensures that the model best represents the current state of knowledge. The current state of knowledge in the case of the model proposed by this chapter is the training set of questions being asked by users querying the underlying data repository.

The solution allows for more questions to be added as the process evolves. The additional questions can be added as part of an automated process or manually. Each question added would need to be tagged and the process retrained. This allows for the continued evolution of the system.

The tagging model used for this solution is the Part of Speech (POS) tagger which converts every word into a token. Each token has an associated tag. OpenNLP will use a probability model to predict the correct tag for each word in the sentence. The fewer the tags used the quicker the performance, this can be seen from testing and appears to be supported by Taghipour and Ng (2015) but more thorough performance testing is required. The tests that were carried out were performed on whole sentences, which included tags that can be identified as having a database related value. An example of this would be where the name of a database table or table column appears in the natural language query. In the case of the natural language query "*Which event has Abdoulaye Diaby played in.*", "*event*" is an identifiable database table. The sentence can then be processed, and relevant tags will be applied to the parts of the query (see Table 5.1), irrelevant tags will be ignored.

The OpenNLP model training task process output: The output from training the model against the grammar file, which contains the list of potential asked questions that is shown in Figure 5.5. Due to the fact that this model is only proof of concept, not much training and comparative analysis was performed on the parameters used. In fact the parameters used for training came from a previous model that had been used for a totally unrelated task. The results from which can be seen in section 5.7 shows that without fine tuning the model can perform well.

```
Indexing events cutoff 5
    Computing event counts... done. 36432 events.
    Indexing… done.
Sorting and merging events... done. Reduce 36432 to 11666 events.
done indexing.
Incorporating index data for training…
done.
    Number of Event Tokens: 11666.
         Number of Outcomes: 3
         Number of Predicates: 2241
done.
Computing model parameters ...
```

Figure 5-.5: The output from the model training process.

As can be seen from the training output, the test was run against a training file with approximately 36,000 entries that were processed and indexed. From the 36,432 source entries, 11,666 were identified as either significant or unique. The number of outcomes in Figure 4.5 refers to the number of possible outcomes from the model. For the shallow parsing approach proposed by this chapter, the number is not significant. Though not significant for this chapter the number of predicates could indicate the number of sentences in the data frame. The predicate identifies what is happening with the subject of a sentence. Though this might be helpful when trying to understand the content or meaning of the sentence for the shallow parse approach being taken by this chapter the number of predicates is inconsequential.

5.7. Evaluation

During the evaluation phase of the proposed system, the idea was to measure the performance of the natural language conversion to SQL. For this chapter the evaluation looks at the speed of conversion from natural language query to SQL and highlights the fact that commodity hardware can be used for the conversion process. Measuring the accuracy of conversion will be tested in the next chapter. The Java Virtual Machine (JVM) usage was monitored, and the code profiled. The details of the proposed system performance are discussed in this section.

5.7.1. Computer System

The computer used for the development and testing of the application is of a standard desktop configuration. The very utilitarian nature of the computer used for developing and testing this solution supports the concept that the conversion process does not require a large, expensive dedicated server. The specifications of the test machine for the natural language to SQL conversion are shown in Table 4.4.

Variable	Value
Operating System	Windows 7 Enterprise
Service pack	SP1
Processor	Intel Core i5-4570 CPU “3.2GHz
Installed Memory	8 GB
System Type	64-bit operating system

Table 5-.4: Server specifications used for testing. The server specifications shown highlights the fact that commodity hardware is suitable for supporting the proposed conversion process.

5.7.2. Java Virtual Machine

The Java Machine used for the development and testing of the application is again a standard build. The application does run on a single JVM instance, the settings for which are shown in Figure 5.6.

The profiling of software allows for some tangible method to measure software excellence as proposed by both Ratnaparkhi (1996) and Siewiorek et al (1993). The tests performed on the software show the resources used for converting a natural language query into a SQL based query. A number of tools have been employed to monitor the performance of the application

which includes Java Visual VM from Oracle, YourKit Java Profiler, and the Coverage tool from JetBrains IntelliJ Java IDE. These tools highlight the computer resources used by the code in terms of virtual memory allocation and call time per function. The concept of benchmarking software performance provides a tangible metric to evidence the performance of a software solution as supported by Sims et al. (2003) .

The benchmarking work carried out by Siewiorek et al. (1993) highlights the fact that monitoring memory is key to understanding the performance of a software solution. The techniques proposed by Siewiorek et al. (1993) and the project findings have been updated by the work of Gama et al. (2011) and Whaley (2000) which also proposed that application memory could also have an important role to play in the performance of an application. In the case of the solution proposed in this chapter the Java Virtual Machine (JVM) is a key component and the memory associated with the JVM is just as important.

```
JVM: OpenJDK 64-Bit Server VM (25.152-b8, mixed mode)

Java: version 1.8.0_152-release, vendor JetBrains

Java Home: c:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2017.3.1\jre64

JVM Flags: <none>

-Xxs24m

-Xxx256m

-Dsun.jvmstst.perdata.syncWaitMs=10000

-Dsun.java2d.noddraw=true

-Dsun.java2d.d3d=false
```

```
-Dnetbeans.keyring.no.master=true

-Djdk.home=C:\Program Files\Java\jdk1.8.0_25

-Dnetbeans.home=C:\Program Files\Java\jdl1.8.0_25\lib\visualvm\platform

-Dnetbeans.user=C:\Users\rskeggs\AppData\Roaming\VisualVM\8u20

-Dnetbeans.default_userdir_root=C:\Users\rskeggs\AppData\Roaming\VisualVM

-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=C:\Users\rskeggs\AppData\Roaming\VisualVM\8u20\var\log\heapdump.hprof

-Dsun.awt.keepWorkingSetOnMinimize=true

-Dnetbeans.dirs=C:\Program Files\Java\jdk1.8.0_25\lib\visualvm\visualvm;C:\Program
Files\Java\jdk1.8.0_25\lib\visualvm\
```

Figure 5-.6: This shows the setting for the Java Virtual Machine on the test server.

The YourKit Java profiler was used to measure the CPU of a conversion from a natural query to SQL. The profile modelled the application through the required classes as part of the execution cycle. Figure 5.6 shows the performance in milliseconds that each class takes to complete a task.

Table 5.6 shows just how much of the code gets executed when converting a simple natural language query to an SQL statement. For the simple example used as part of the test the execution time to convert the natural language query to SQL took a total of 665 milliseconds.

The Java Visual VM tool provides detailed information about Java applications while being executed on a Java Virtual Machine. The performance figures highlight the fact that no specialist hardware is required to run the process, which could be hosted on commodity hardware. To substantiate this table 5.6 shows the results from the Visual Machine usage, that the largest resource allocation during testing was 42 Mb which accounted for 51% of all memory allocations by the virtual machine. Running tests against larger data will use more resources but the need to move to specialist hardware may not be a requirement, though further testing will need to be conducted to determine more accurately resource requirements. Tuning for performance in high throughput environments can also be managed by distributing resources across a platform when bottlenecks are identified. More in depth testing will need to be carried out to understand where and when these limits are reached. Figure 5.8 shows the memory usage of the conversion process used when converting a simple string like "*What are the odds on a game involving abdoulaye diaby?*" into the SQL statement "*select * from event where player = 'abdoulaye diaby*".

Having completed a conversion and extraction of data from the dataset the next step was to compare performance of the system discussed in this chapter with other comparable systems. For this, the paper by Joshi and Akerkar (2008) proposed a similar approach using a Part of Speech based algorithm for converting natural language into an extraction-based query. The researchers compared the performance for two systems and the results are summarised in Table 5.5.

Type of Data	No of words	Time Required by QTAG (Used in Enlight)	Time Required by Minipar (Used in Sapere)
Times of India	202	1.71 secs	2.88 secs
Reply START QASystem (251Words) University Information	251	3.11 secs	
NMU Broadcaster	226	1.55 secs	2.86 secs
Wikipedia	226	1.67 secs	3.13 secs
Average		1.705 secs	2.9925 secs

Table 5-.5: Shows the performance figures from the Joshi and Akerkar (2007) paper.

Call Tree		Time (ms)	%
All threads		665	100
nlidbPOSNLIDB.main		509	77
POSNLIDB.java	nlidb.POSNLIDB.translate	156	23
POSNLIDB.java	nlidb.POSNLIDB.tokenizer	156	23
NLTokenizer.java	opennlp.tools.tokenizer.TokenizerModel	124	19
NLTokenizer.java		31	5

Table 5-.6: Shows the execution time the conversion process takes through the components of the conversion code. The data comes from YourKit Java Profiler. The screenshot is in Appendix D.

The main figures to take away from table 5.6 is that the whole processing time was 665 milliseconds, and the nlidb.POSNLIDB library used 509 milliseconds.

Allocated Object	Bytes Allocated	Objects Allocated
int[]	42355328	501458
char[]	13223970	243009
java.lang.String	3141200	130887

java.util.HashMap	2043264	63852
java.nio.HeapCharBuffer	1691904	35241
java.langLong	1626240	67750
jazva.lang.Object[]	1238920	22331

Table 5-.7: Shows the memory allocation for the conversion process. Original screenshot is shown in Appendix E

Tables 5.7 and 5.6 highlight the fact that the compute resources required to run a conversion process to translate a natural language statement to a language capable of querying a database does not require a large amount of resources, either in terms of memory or CPU. main figure to take away from the screenshot in figure 5.8 is that the bulk of the resourcing required memory for an int array (42355320 bytes) and char array (13223976 bytes)

The paper by Joshi and Akerkar (2008) did not specify the specification of the computer used to carry out the benchmark. The questions used by the Joshi and Akerkar (2008) paper were taken from the TREC-2005 Question Database but there was some ambiguity in identifying the actual datasets used for the benchmarking. In comparison, this chapter has taken a much larger dataset and has added the additional complexity of creating a join between two tables. The natural language questions used by this chapter were of a similar complexity to the questions used in testing carried out by Joshi, Akerkar (2008) and are listed in Figure 5.8.

The use of an older computer configuration as can be seen from Table 5.4 should be more comparable to the server used by Joshi and Akerkar (2008) in their paper. This makes the comparison between the two papers more about the performance of the software than the hardware. The average conversion time using the solution proposed by Joshi et al (2008). was 1.7 seconds with the fastest being 1.5 seconds.

Testing the solution proposed by this chapter the conversion time from natural language to structured query language took consistently under 700 milliseconds. The datasets from this chapter consists of two files one containing over 36,000 events and the other over 11,000 (see Figure 5.5). Were also larger than the datasets used by Joshi et al (2008). as these datasets contained approximately 220 records (see Table 5.5). Table 5.5 also shows the completion of time for the solution proposed by Joshi et al. (2008) and Table 4.7 also contains the times of each process to complete by the solution discussed in this chapter. In summary, the tables highlight the improvements in performance the approach being taken by the thesis as compared to other existing solutions.

- Who killed militants?
- Who did Forman defeat for his first heavyweight championship?

- What do frogs eat?
- Who visited Bill Clinton?
- Who did France beat for the World Cup?
- What is the largest volcano in the solar system?
- What is the longest river in the world?

Figure 5-.7: Sample questions used for performance comparison by Joshi, AkerKer (2007).

Collection	Number of Words	SQL Conversion	Data Extraction
ginf.csv	19531	0.665 secs	0.9 secs

Table 5-.8: Performance from the proposed system which includes the conversion from natural language to SQL.

5.8. Conclusions

There are a number of limitations to the system being proposed in this chapter. The storage space required for the grammar file and index file might make this solution unworkable. More testing against larger datasets is also required to understand the limitations and performance of the proposed solution. This chapter has suggested a solution for joining tables together. Further testing would also be required to validate the performance of joining more than two tables.

The biggest issue that has not been addressed by this chapter is around the selection of data points being retrieved from the underlying database. Currently, the solution relies on the statement SELECT * which retrieves all data points from the tables being searched. Retrieving data from all columns in the target database could prove to be costly in terms of memory and processing resources. Refining the SELECT statement could possibly be achieved through the use of deep learning techniques. It may be possible to identify columns in tables that have a higher probability of being selected.

Regardless of the identifiable shortcomings from the proposed system, the thesis has reinforced the benefits of using part of speech within a framework that translates natural language into a query language for searching a database. Performance of NLIDB solutions has been an issue that researchers are continually trying to improve upon the performance of NLIDB based solutions. According to the likes of Florin et al. (2017), Gallant (2019), Joshi et al (2008), Voorhees (2001) the common performance issues are speed and accuracy of conversion. As can be seen from this chapter the performance of the proposed system is an improvement in speed when compared against the performance recorded by Joshi and Akerkar (2008) as recorded in Table 5.6.

The question posed at the beginning of this chapter revolved around the requirement to understand nuance when converting natural language to a language capable of querying a database. This chapter shows that the shallow nature of the parsing through the use of the natural language part of speech also reduces the need to understand the complexity underpinning language nuance. This has been highlighted by the steps required to take a natural language statement and produce a query capable of querying a database. The results from the speed of conversion as shown in Section 5.7 shows the improvement in performance without understanding language nuance. The next chapter continues the use of shallow parsing that highlights not only the increase in the speed of conversion but also highlights an improvement to accuracy. The improvement in performance stems from the use of machine learning techniques within a sequence-to-sequence model.

6. Improving The Shallow Parsing Approach

6.1. Introduction

The ideas in this chapter expand on papers presented at the EFiC and IDA conferences in 2017. Converting natural language into a structured query language (SQL) should be as straightforward as translating text from one natural language to another such as French or Spanish. SQL is just another language with its own syntax and grammar. The fact that work has been ongoing in this field since the LADDER project of the 1960's shows that the conversion from Natural Language to SQL is not simplistic. This chapter expands on the concepts from the previous chapter and introduces a novel approach to solving this problem.

The idea proposed in this chapter combines the use of sequence-to-sequence models in conjunction with the natural language part of speech technologies and domain specific languages to convert natural language queries into SQL. The approach being proposed by this chapter is to use natural language processing to perform an initial shallow pass of the incoming query and then use Google's Tensor Flow to refine the query with the use of a sequence-to-sequence model. The thesis is also proposing to use a Domain Specific Language (DSL) as part of the conversion process. The use of the DSL has the potential to allow the natural language query to be translated into more than just an SQL statement, but any query language such as NoSQL or XQuery.

Natural Language into Database (NLIDB) has been within the research lexicon for a number of years. Early systems such as LADDER, PRECISE, NaLIX and WASP emerged from the research community but failed to make any major impression within industry. A recent literature review by Ahkouk et al (2019) has highlighted the lack of uptake by the commercial software vendors was based on the poor performance of the translation from natural to SQL. The performance issues can be categorised into two broad based areas. The first of these is how to handle the nuance of language in the conversion from natural language to SQL. This point has been identified by the likes of Voorhees et al (2001) and Bais et al (2018). The second of these issues is based on the accuracy of the conversion from natural language to SQL through lack of understanding the underlying database. This has previously been highlighted by the work of both Joshi et al (2008) and Skeggs et al (2019).

This chapter proposes a solution to solve both the language nuance that was highlighted by both Voorhees et al (2001) and Bais et al (2018) and performance issues with accuracy of converting natural language to SQL as shown by Joshi et al (2008). The approach being proposed is to use shallow parsing as this does not require an under-standing of language nuances, it identifies keywords in the input text as discussed in the paper of Ratnaparkhi and Adwait (1996). These keywords are used to identify characteristics in the input statement that are important for the search. Jwalapuram and Mamidi (2007) are among a number of authors who have carried out research into using keywords to enable NLIDB based systems to perform searches. The keyword searching proposed in this chapter is built from the underlying database unlike that proposed by Jwalapuram & Mamidi (2007). Both Jwalapuram & Mamidi (2007) and this chapter propose using a Part of Speech (POS) tagger but this chapter is also proposing to use a sequence to sequence model to refine the translation to SQL.

To support the testing and validation of the work being discussed in this chapter, the process is tested against an AirBnB dataset. The rational for using this dataset is that it is used by the WikiSQL project and has a process for marking the output for the text to SQL conversion process. There are few datasets available that allow for the same type of validation as the datasets found in the WikiSQL project.

6.2. Related Work

Most of the work related to this project have either taken an approach that has relied on semantic parsing or on the use of sequence-to-sequence models. Some projects like the approach being proposed by this chapter have taken a multi-step approach. The sections below discuss the work within semantic parsing and sequence to sequence that relate to this chapter along with the use of combining multiple steps and technologies to convert natural language to structured query language.

6.2.1. Semantic Parsing

The concept of semantic parsing in its simplest form is taking a natural language statement and converting it to a logical form that is machine understandable. Lin et al (2017) take this concept in its purest form to convert natural language to bash. This chapter also highlights that it is not just SQL being used as a target for converting natural language to machine capable language. This chapter concentrates on the use of SQL as a target language. The research by Shah et al (2020) is based on speech to SQL and as part of their solution they propose a new language SpeakQL. They also create a dataset specifically for speech-based SQL conversions. The approach used by Shah et al (2020) relies on semantic parsing for the creation of the SQL statement. Research into the use of semantic parsing can be classified under the following headings: executed guidance, tree structures, underlying database structured, descriptive language and user interactions.

6.2.2. Executed Guidance

The first of these executed guidance uses statistical analysis to select the best output from a number of possible solutions. This approach originates from the work carried out by Wang et al (2018). Their concept looks at statements in various stages during the conversion process and discards those statements that cannot complete the conversion to SQL. Yin, Neubig (2019) take a similar approach to Wang by ranking the predicted output from the conversion model selecting those with the highest score. Talmor and Berent (2018) take this one step further by using the internet as their model for training.

6.2.3. Tree Structures

Use of tree structures for solving the problem with semantic parsing has been used by both Cheng et al (2018) and Yin et al (2018). In the case of Cheng their work uses the tree structure with a domain grammar to ensure that the conversion is accurate. In contrast Yin et al use tree structures to hold the training data which can be labelled or unlabelled.

6.2.4. Underlying Database Structure

Karki et al (2019) and Bogin et al (2019) both rely on the underlying database structure as part of the process in parsing the natural language statement. Bogin et al (2019) model the database structure within a graph schema as a method of understanding the relationships between tables. In contrast Karki et al (2019) construct a row and column based grid from the database features.

6.2.5. Descriptive Language

The use of a descriptive language can also be used with the semantic parser. In the case of Yin and Neubig (2018) they propose using abstract syntax description language for parsing the natural language onto an SQL template. Lin et al (2019) use a schema dependent grammar to map the natural language onto a SQL syntax. Campagna et al (2019) look at using a Virtual Assistant Programming Language (VAPL) to formalise the natural language statement. Cheng et al (2019) take a similar approach to Campagna et al (2019) as they use templates that can map the text from the natural language onto an SQL structured template.

6.2.6. Sequence to Sequence Models

In contrast to semantic parsing the concept of a sequence-to-sequence model is to train a model to take a sequence from one domain or language and convert to another domain or language. These models have been used to convert text from one natural language to another and have now been introduced to convert from a natural language to a programming language such as *bash* in the case of Lin et al (2018) or *SQL* as proposed by Shi et al 2018.

Recent research using sequence to sequence models has extended the approach typically used by Lin et al (2018) which takes a natural language statement and then compares it to a *bash* command. In short a sequence to sequence model is a type of Encoder-Decoder model using recurrent neural networks (RNN). From the research undertaken in this field a Sequence-to-sequence research project can be categorised into two streams.

- Models that have begun using the content from the database rather than the database structure to understand the structure of the data,
- Models that are chunking the natural language statement into smaller more discrete blocks to create multiple seq2seq models.

In the model proposed by Shi et al 2018 their work uses a sequence to action model. The solution uses a SQL template with place holders to contain the name of the table, columns and variables. Sequence to action models are then used to parse the appropriate values into the template. With this project the underlying database structure is central to the conversion process.

There are also extensions to the traditional sequence to sequence model such as the work carried out by Xu et al (2018) which uses a graph based neural network to create a graph to sequence model. The work carried out by Yu et al (2018) uses a tree network to create what they refer to as a text-to-SQL model. Wang , Tian et al (2018) also take a similar approach with their text to SQL model. Wang instead proposes separating the data from the schema within the sequence model. Guo et al (2019) also propose a text-to-SQL model by creating a multiple step approach to the problem of converting natural language to SQL. As part of the process the solution creates a synthetic query language from the natural language statement and database structure. The final query is inferred from the synthetic query.

In parallel to the extension of the sequence-to-sequence model the more traditional sequence to sequence model is being refined by the likes of Soru et al (2018) which like Xu et al (2018) uses graph patterns to learn the sequence make up of relationships between elements. The work by Soru though is more of a traditional sequence to sequence approach. In comparison Su et al (2018) propose using multiple sequence to sequence models at each step along the process of conversion. They also support the use of user interaction to correct errors in the process of converting natural language to SQL.

Part of a sequence-to-sequence solution relies on the decoding or the translation from the input to the output. Bello et al (2018) assigns an item score as part of the decoding process. The score is based on historical data, and according to Bello allows for *higher-order interactions*. In contrast Zavershynskyi et al 2018 use a multiplicative attention mechanism as part of the RNN within the decoder.

The work performed by Guo and Gao (2018) like Su et al (2018) chunked the natural language statement into smaller elements thereby creating a chain of sequence-to-sequence models. Within the *WHERE* clause the team ranked possible solutions based on historic data to choose the option with the highest ranking score.

Other approaches for handling sequence to sequence models like Petrovski et al (2018). The team proposes removing the database structure completely from the sequence-to-sequence model and relying solely on the content of the tables to describe the content of the database table. Then Sabour et al (2019) were more concerned with the method of training the sequence-to-sequence model. The approach they propose was to create an *Optimal Completion Distillation (OCD)*. This required statistically sampling the data used for training based on predefined characteristics.

The Python TensorFlow library provides a sequence-to-sequence library which is employed by this project to build the model. The sequence-to-sequence model was not and does not need to be a highly refined custom model. This project, even though it is a proof-of-concept project, wants to show that sequence to sequence models could be used in the context of a natural language to structured query language domain capable of successfully searching a database.

6.2.7. Multi Step Architecture

Sequence to sequence models and semantic parsing are two approaches that concentrate on a single part of the process required to convert natural language to a structured query language. Few like the approach being proposed in this chapter have taken a multi step approach to refining the process of converting natural language to SQL. Polosukhinet et al

(2018) use a multi step approach within their work which takes a similar approach to that being proposed by this chapter. Both this chapter and the work by Polosukhinet et al (2018) use a domain specific language (DSL) as part of the conversion process. In contrast the work carried out by Polosukhinet et al (2018) use an extension to the sequence-to-sequence model that they refer to as Seq2Tree whereas the solution being proposed by this chapter uses a more standard version of the sequence to sequence model as proposed by Soru et al (2018).

Likewise Lukovnikov et al 2018 use a combination of augmented pointer along with LSTM *column encoders*, and a sequence to sequence model in conjunction with semantic parsing to translate the natural language statement into a query language. Taking a similar approach is Choi et al (2020) again using sketching like the work carried out by Zhang et al (2020) to extract the pertinent data from the natural language input statement which can then be transposed on the SQL template. Unlike the work being proposed by this chapter they also propose recursively predicting nested statements.

In Joshi et al (2020) take a hybrid approach to the conversion of text to SQL. Their work uses a series of sequence-to-sequence models to create the SQL statement they also propose user interaction. Unlike Gur et al 2018 who propose user interaction to refine the process, Joshi et al (2020) are after restricting the inputs from users to avoid linguistic variations and ambiguities in the statement.

6.3. The Model

The model being proposed by this chapter uses a multi architectural approach similar to Polosukhinet et al (2018) encompassing a number of technologies in the pursuit of converting a natural language query into an SQL statement. The approach uses not just sequence to sequence models but also domain specific languages and shallow syntactic parsing. Natural Language Parsing is used to identify important features in the input statement. Having identified the features, the detail is wrapped into a JSON object for storage and ultimate transformation to SQL. Figure 6.1 shows the flow of data through the proposed system highlighting the steps that the input natural query statement undergoes as part of the translation to structured query language. This process differ slightly from the previous section in that a sequence to sequence model has now been introduced into the flow and the parse process that finally creates the SQL statement has been refined into an internal light DSL. This section discusses the process in detail.

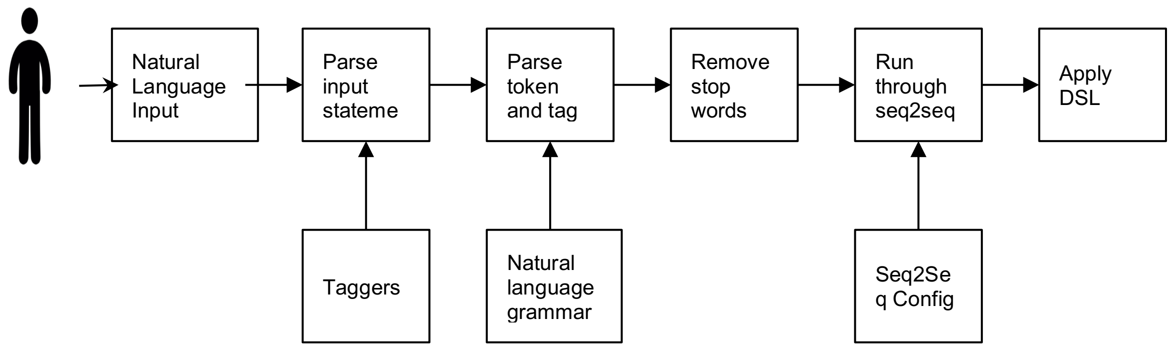


Figure 6-.1: The diagram shows the flow of data through the proposed system. The sequence to sequence model is introduced to the conversion process. The input is now the DSL proposed by this chapter.

The processing steps through the system are:

- The input into the process is the proposed DSL. Taking this approached reduces the amount of configuration required for the parsing and tagging steps. It may also be possible to remove the step that takes out the stop words, though more testing is required.
- Parse the input statement into tokens: The natural language query is broken into word tokens using the appropriate tagger. This chapter proposes the use of two taggers OpenNLP Part of Speech tagger and the OpenNLP Names Tagger. Both taggers can be used in parallel each to perform the task of identifying key words that are useful to the conversion process and to identify names that are within the input statement.
- Parse Tokens and tag. The next step is to take the word tokens which is the output from the previous step and apply a tag. The grammar file contains the details of the tags to be applied to each token.
- Remove stop words. The next step is to remove the stop words from the process as these words add nothing to the conversion process. This step potentially redundant with the use of the DSL as an input.
- Run untagged words through sequence to sequence. Then any words that remain untagged can be run through the TensorFlow sequence to sequence model and appropriate tags can be applied to those word tokens.
- Apply DSL. The proposal is to use an internal light DSL which will take the data structure that contains all the tagged data and create the SQL statement. The idea

for the internal DSL is that the output from this step could in fact be any query language not necessarily SQL.

6.3.1. Natural Language Processing

The Natural Language toolkit used by this chapter is the NLTK Python based NLP toolkit developed by Bird et al (2008). The advantage of using this library with Python was the speed at which a proof of concept application could be developed. The idea behind the use of natural language processing in this chapter is for the identification of important components within the natural language input to facilitate the conversion to structured query language. This chapter uses *part of speech* and *name* tagging to extract pertinent data from the input query. Having first specified elements which are important for the processing step as shown in code sample 6.1 and code sample 6.2. The structure used stores the keywords and associated tag as a Python list. The first element in the list is the keyword or identifying regular expression followed by the tag.

As part of the NLP process the input statement is tokenised with each word being extracted using the space character as identifying when a word ends. Known elements are extracted from the input query and tagged before being stored as a JSON object. Code sample 6.1 shows how a regular expression can be used as part of the tagging process. The regular expression statement shown in **code sample 6.1** identifies postcodes within the input query.

```
patterns= [
(r’( [ A-Za-z ] [ A-Za-z ] ? [0-9] [0-9] ? [ \s ] ? [ A-Za-z ] ? [0-9] [0-9] ? [ A-Za-z ]
[ A-Za -z])
(r’( [ A-Za-z ] [ A-Za-z ] ? [0-9] [0-9] ?)’,’out’),
(r’( [ A-Za-z ] ? [0-9] [0-9] ? [ A-Za-z ] [ A-Za-z]),’in’))
```

Code Sample 6-.1: This code extract shows a potential solution to how postcodes could be handled with the use of regular expressions. It also shows how a regular expression can be used within the NLP tagging process.

The code extract shown in code sample 6.2 shows a more conventional approach to setting up a NLP tagger within an NLP process. A value extracted from the underlying database is tagged with the column name from where it came. The tag is later used within the SQL statement as part of the select statement.

```
names =[
( ‘Chris’ , ‘firstname’)
]
```

Code Sample 6-.2: NLP tagger grammar construct for use in extracting names.

The grammar construct is used by the tagger function shown in code sample 6.3 to identify each individual word in the input statement. The tagger function takes three parameters:

● **tokens:** This is each word from the input query as a list of tokens. The tokens are created from the tokeniser function by splitting the input statement on the space character.

● **patterns:** This is the list of predetermined keywords with associated tags. This is the first of the taggers

● **names:** This is the list of predetermined keywords with associated tags. This is the second of the taggers and in the example, it is used as the back-off tagger. A back-off tagger comes into its own when the primary tagger fails to identify a word for tagging.

```
def Tagger (tokens, patterns, names):
    name_tagger = nltk.RegexpNameTagger(names)
    regexp_tagger = nltk.RegexpTagger(patterns, backoff=name_tagger)
    tagged = regexp_tagger.tag(tokens)

    for x in range(len(tagged)):
        if tagged[x][1] =='out' and tagged[1+x][1] = 'in':
            tagged.append (tagged[x][1] + ' ' +  tagged[1+x][1],
'postcode')

    return tagged
```

Code Sample 6-.3: A simple Python function that will tag an input statement and create a JSON construct to contain each word token with the appropriate NLP tag. The function takes 2 taggers.

Code Sample 6.3: A simple Python function that will tag an input statement and create a JSON construct to contain each word token with the appropriate NLP tag. The function takes 2 taggers.

The returned value from the function shown in Table 5.3 is the chunked input statement tagged with the appropriate NLP tag stored as a JSON object. From the following example a simple natural language input query 'which Chris lives in the area EC2A 5AP. The output from the tagging function is shown in Table 5.4.

```
[('which', 'None'), ('Chris', 'firstname'),  ('lives', 'None'), ('in',
'None'),  ('the', 'None' ), ( 'area', 'None'), ( 'EC2A 5AP', 'postcode')]
```

Code Sample 6-.4: JSON output from the simple input query.

Words from the input query that cannot be identified by the tagging processes are tagged with None. The untagged words can still be useful for the transformation to SQL. Next, stop words are removed from the list of untagged words. The final step is then used by the Sequence-to-Sequence Model (section 6.3) to identify which words can be identified and tagged as having a relevance to the underlying database and should be part of the query.

6.3.2. Internal Domain Specific Language

Having identified which words within the input statement are pertinent to the conversion from natural language query to SQL statement. Expanding on the parse process in the previous chapter the process has been refined and now resembles a domain specific language. Dursen et al define a Domain Specific Language as a "small declarative language that offers expressive power focused on a particular problem domain". The particular domain that this chapter is concerned with is converting JSON (JavaScript Object Notation) to SQL. The current process will need to be refined further for it to be a pure domain specific language implementation.

During the development phase of the internal domain specific language, it was noticed that the target output could potentially be another language such as Xquery or NoSQL. More work will be required to complete the internal DSL with further work required to target other languages.

6.3.3. Sequence to Sequence Models

Sequence to sequence models are currently being used in speech recognition systems as shown by the work carried by Chui (2018) as well as in language translation scenarios using neural networks which as proposed by Weiss et al (2017). The paper by Weiss et al (2017) highlights their use in language translation as it proposes a solution for translating Spanish text to English. The same principles can be applied in the translation from English to Spanish as English to SQL. Structured Query Language is after all just another language but one that is designed to work with databases as described in ISO/IEC 9075-2:2016.

This chapter proposes taking these already defined use cases for sequence-to-sequence modelling and applying them to translate a natural language into an SQL statement capable of querying a database. In this chapter the first sequence of words is the natural language input and the second sequence is the equivalent SQL translation.

We propose using the machine learning components within TensorFlow to create the Sequence-to-Sequence model. Both Chung-Cheng Chui et al (2018) and Yaser et al (2019) propose using sequence to sequence models for translating one natural language into another natural language, proposing that the approach improves accuracy of translation. The researchers have not considered the application of this approach for use with a query language like SQL, and the improvement in accuracy it gives. In this section we look at the implementation of the following approach.

The sequence-to-sequence model first takes the words that were not successfully tagged as part of the natural language process defined in section 3.1 and were not identified as stop words. These words are then compared to the translation file, an extract of which is shown in code sample 6.5. Associated with each input statement is a corresponding sequence which is used to build up the final SQL statement. From code sample 6.5 the first column shows the words from the input statement which are to be translated (area, name, address). The second column contains the appropriate translation to be used. In the case of the word area the translation is tbl_customer, address.

```
area    tbl_customer, address
name    tbl_customer, firstname surname
address    tbl_customer, house_name_number first_line town city postcode
```

Code Sample 6-.5: The table shows the content of the tab separated sequence to sequence translation file.

The entries in the second column (tbl_customer, address) are database components. Within this chapter the first element tbl_customer is the name of the database table, the second element address is the column within the table that could be applied as part of the translation for area.

The sequence-to-sequence model requires a data definition file that takes the expected input and the corresponding output file. Sequence to Sequence (seq2seq) models typically use Recurrent Neural Network (RNN) architectures to solve language problems like machine translation and chatbots. The sequence-to-sequence model being used in this thesis falls into the machine translation grouping as it takes a natural language input and converts to a machine language output. The approach being used in this thesis differs from the normal approach which is to convert from one natural language to another. Like most sequence-to-sequence models the approach being used in this thesis is to use an encoder-decoder model.

The sequence-to-sequence model requires a dictionary of terms, this is an expected input and a corresponding expected output. In the case of this project the input is a series of natural language words and the expected output is a series of database objects that can be extracted from the underlying database. These database objects are database table names and table attributes (column names).

Model Input	Model output
['area',	['ab_nyc_2019, neighbourhood_group',
'room',	'ab_nyc_2019, room_type',
'accomodation',	'ab_nyc_2019, room_type',
'address',	'ab_nyc_2019, neighbourhood_group',
'district',	'ab_nyc_2019, neighbourhood_group',
'city',	'organisations, city ',
'country',	'organisations, country ',
'organisation',	'organisations, org_name ',
'person',	'contacts, lastname',
'school',	'sports_clubs, school_name',
'club',	'sports_clubs, club_name',
'team']	'sports_club, team_name']

Table 6-.1: An extract from a data dictionary used in this thesis to highlight the use of a sequence to sequence model. This is based on the data from code sample 6.5 but represented as two lists of data.

The content of code sample 6.5 cannot be generated initially through an automated script. Identifying the database table and the associated database column can be automated. Matching the keyword with the appropriate database table and column is not currently possible to automate. The process for building up the file in this chapter was a manual process. The design and testing of the model should allow for keywords to be identified and matched with the underlying database table and table attributes. Using production queries to identify and match keywords with database components should be encouraged.

The sequence-to-sequence model used for this chapter was a small sequence model published in a python reference book State of-the-Art Speech Recognition with Sequence-to-Sequence Models. Part of the logic for choosing this particular algorithm was to emphasise that a simplistic sequence to sequence model could be used as part of an architecture for translating natural language into SQL. The sequence-to-sequence algorithm was originally designed and used to demonstrate how it could act as a chat-bot. With the addition of the natural language tagger from section 3.1 and the domain specific language described in section 3.2 this chapter shows how the same algorithm can be applied to the translation of a natural language statement to SQL.

The model used in this project was created using Google's TensorFlow. The Figure 6.2 is a pseudo code representation of the actual python model. Within the pseudo code example the data to be modelled is split between the input which is extracted from the incoming natural language statement and the output which is the data from the dictionary or in the case of Table the model output.

```
Seq2SeqModel
    #intialise a number of terraform placeholders that will contain the
data for the sequence to sequence model.
    X = placeholder
    Y = placeholder
    X_len = placeholder
    Y_len = placeholder

    # for the encoder
    Create a randomised tensor
    create a look-up for the tensor
    Add an index to the data

     # for the decoder
    Create a randomised tensor
    create a look-up for the tensor
    Add an index to the data

    # for the encoder
    Create the RNN cell

    # for the decoder
    Create the RNN cell

    create logits for the softmax function

    Use the Adam optimizer for training.
```

Code Sample 6-.6: A pseudo representation of the sequence to sequence model used in his project.

Tensors are created for both the encoder which is the input data to the model and the decoder which is the output from the model. A tensor lookup and an index are also created for both the encoder and the decoder. Multi RNN cells are created for both the encoder and the decoder. A logits is also used before using the Adam optimizer.

The Sequence-to-Sequence model within this chapter uses a long short-term memory (LSTM) recurrent neural network (RNN). The internal memory of the LSTM network is particularly useful for the processing of sequence data. Prabhavalkar1 et al (2017) and

Chung et al (2014) both support the concept of using RNN within a Sequence-to-Sequence model, for the performance benefit over other neural network models. Prabhavalkar1 et al (2017) also states that increasing the numbers of layers in the decoder increases the performance of the model by up to 7%. Their paper does not identify a recommendation of the optimal number of layers nor does it recommend what the upper limit on number of layers should be. The number of layers that are being used by this chapter are two which is highlighted in Table 6.6. Setting the number of layers to two ensured that this proof of concept model was not optimally tuned and would also reduce the amount of time required to run the model.

The python code for testing the model is a simple script that follows the following steps.

- Reset the tensor flow DAG.
- Create a tensorflow interactive session.
- Pass the parameters into the tensorflow model.
- Bookmark and save model.

The actual code used for testing is shown in code sample 6.7.

```
tf.reset_default_graph()
sess = tf.InteractiveSession()
model = seq2seq(size_layer, num_layers, embedded_size,
vocabulary_size_from + 4,
                vocabulary_size_to + 4, learning_rate, batch_size)

sess.run(tf.global_variables_initializer())

saver = tf.train.Saver(tf.global_variables(), max_to_keep=2)
checkpoint_dir = os.path.abspath(os.path.join('./',
"checkpoints_chatbot"))
checkpoint_prefix = os.path.join(checkpoint_dir, "model")
```

Code Sample 6-.7: The simple python code used to train the model.

The code sample 6.8 shows a simple testing script written in Python to test the accuracy of the model.

```
for i in range(epoch):
    total_loss, total_accuracy = 0, 0
    for k in range(0, (len(text_from) // batch_size) * batch_size,
batch_size):
        batch_x, seq_x = pad_sentence_batch(X[k: k+batch_size], PAD)
        batch_y, seq_y = pad_sentence_batch(Y[k: k+batch_size], PAD)
        predicted, loss, _ = sess.run([tf.argmax(model.logits,2),
model.cost, model.optimizer],
                                      feed_dict={model.X:batch_x,
                                                model.Y:batch_y,
                                                model.X_seq_len:seq_x,
                                                model.Y_seq_len:seq_y})

        total_loss += loss
        total_accuracy += check_accuracy(predicted,batch_y)

    total_loss = (len(text_from) / batch_size)
    total_accuracy = (len(text_from) / batch_size)
    print('epoch: %d, avg loss: %f, avg accuracy: %f'%(i+1, total_loss,
total_accuracy))
    path = saver.save(sess, checkpoint_prefix, global_step=i+1)


    total_loss = (len(text_from) / batch_size)
    total_accuracy = (len(text_from) / batch_size)
```

Code Sample 6-.8: A simple python script to test the accuracy of the model.

The output from the test script is.

```
epoch: 1, avg loss: 0.156250, avg accuracy: 0.156250
```

The value for the *epoch* is a parameter that is passed into the model as can be seen from Table 6.2. The average loss value at 0.15 and an accuracy of 0.15 has little meaning from the sample set used as the sample set is too small to produce accurate values.

The sample python script in code sample 6.9 shows how the word passed into the function calls the model to produce an output. Taking the word '*area'* from the dictionary in Table 6.1, the return value from the model is *'neighbourhood_group'*.

```
def predict(sentence, rev_dictionary_to):

    X_in = []
    for word in sentence.split():
        try:
            X_in.append(dictionary_from[word])
        except:
            X_in.append(PAD)
            pass


    test, seq_x = pad_sentence_batch([X_in], 1)#PAD

    input_batch = np.zeros([batch_size,seq_x[0]])
    input_batch[0] =test[0]

    log = sess.run(tf.argmax(model.logits,2),
                                      feed_dict={
                                              model.X:input_batch,
                                              model.X_seq_len:seq_x,
                                              model.Y_seq_len:seq_x
                                              }
                                      )

    result=' '.join(rev_dictionary_to[i] for i in log[0])
    return result
```

Code Sample 6-.9: A simple function that is used as an entry point to the sequence to sequence function.

The parameter values used for this model are shown in code sample 6.8. There was no performance tuning carried out to optimise the performance of this model. The values were chosen based on experience working with other similar models.

VARIABLE	VALUES	DESCRIPTION
SIZE_LAYER	128	Long short-term memory unit (LSTM) recurrent network cell. The number of units in the LSTM cell.
NUM_LAYERS	2	RNN cell composed sequentially of a number of multiple simple cells.
EMBEDDED_SIZ E	128	The generated values follow a uniform distribution in the given range
LEANING_RATE	0.001	The default value of is 0.001 is a highly recommended default value [27]
BATCH_SIZE	32	defines the number of samples to work through before updating the internal model parameters
EPOCH	1	The number of times the algorithm is going to run

Table 6-.2: Shows the value of the neural network parameters used by this chapter. The parameter values have not been changed from the original model.

The parameter values used for the model were selected based on the parameters used by another model that had previously been developed to identify certain words in a streamed dataset. From previous experience working with sequence-to-sequence models it was believed that the parameters chosen would provide a satisfactory response for a proof of concept scenario. Further refinement of the parameter values may provide improved performance in both speed and accuracy of conversion. This model refinement was not part of the original project scope. The idea behind these settings was to deliberately use parameter values that were not optimised.

For the model used in this chapter the epoch was set deliberately low and not just to improve the speed at which the model is trained but to also show that the model was effective with minimal training. With the epoch being set to 1 the training time was only 0.001 seconds. A side effect of having such a low epoch is to highlight in the simple example of this chapter the accuracy of the sequence-to-sequence translation is still high which is discussed in section 6.4.

The training values from the algorithm are epoch 1, average loss 0.093750 and average accuracy of 0.093750. With an epoch of 1 and a batch size of 32 reading too much into the values of loss and accuracy should be discouraged. Parsing name and area into the Tensor

Flow sequence to sequence model the output is as expected, area returns the table tbl_customer and the attribute address. The input of name returns the table tbl_customer with the attributes firstname and surname.

Putting all the pieces together, the algorithm takes the natural language input statement of '*Chris lives in the area EC2E 5BR?'*. The input statement is passed to the model and the parameters from the input query make up the requested parameters from the select statement. Then using the details from the sequence-to-sequence model details of the table to be queried and conditions of the query are constructed. The returned SQL statement from the model is,

> *select address, house_name_no, first_line, town, city, country, postcode, firstname, surname where firstname like 'Chris' or out like 'EC2E' or in like '5BR' or postcode like 'EC2E 5BR'*

The entire process from input statement to SQL query took 7 seconds which also includes the training time for the model on a laptop running OS Name Microsoft Windows 10 Pro Version 10.0.18362 Build 18362. The processor is Intel(R) Core(TM) i57200U CPU @ 2.50GHz, 2712 Mhz, 2 Core(s), 4 Logical Processor(s) running 8Gb RAM.

6.4. Validating the Model

Having developed the approach against a small dataset the next step was to validate the model against a larger dataset. To test the approach being proposed by this chapter we used the New York City Airbnb data available from Kaggle

(https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data/data) as the underlying dataset. The data set has 49000 entries and 16 columns in a single CSV file, the meta data for the dataset is shown in Appendix B. Using the same preparation as discussed in section 3 the first step is to build a tagging model for the data. A sample configuration for tagging the words is shown in code sample 6.10, the values are taken from the underlying dataset which in the case of the New York City AirBnB dataset is the column names, therefore we have tags such as *firstname*, *neighbourhood_group* and *room_type*.

```
names =[
('Chris','firstname'),
('Brooklyn','neighbourhood_group'),
('Manhattan', 'neighbourhood_group'),
('Queens', 'neighbourhood_group'),
('Staten Island', 'neighbourhood_group'),
('Bronx', 'neighbourhood_group'),
('Private Room', 'room_type'),
('Entire Home/Apt', 'room_type'),
('Shared Room', 'room_type'),
]
```

Code Sample 6-.10: The content of the NLP tagger.

With the NLP tagging configuration shown in **code sample 6.10** the natural language query *looking for a room in Brooklyn* can be tagged, the output of which is shown in **code sample 6.11**. The tagging phase only picks up the name *Brooklyn* the rest of the words are labelled as *None*.

```
[(looking, 'None'),  ('for', 'None'),  ('a',  'None'), ('room', 'None')
('in', 'None') , ('Brooklyn', 'neighbourhood_group')]
```

Code Sample 6-.11: Shows the output from the NLP tagger.

The content of the JSON object is just {"neighbourhood_group":"Brooklyn"}. Next step is to drop stop words from the input state statement which just leaves the word *room*. From the sequence-to-sequence model an extract of which is shown in **figure 6.2.** it can be determined that the word *room* will be mapped to *room_type* which is a column in table *AB_NYC_2019*.

```
area              AB_NYC_2019, neighbourhood_group
room              AB_NYC_2019, room_type
accommodation     AB_NYC_2019, room_type
address           AB_NYC_2019, neighbourhood_group
district          AB_NYC_2019, neighbourhood_group
```

Figure 6-.2: The content of the Sequence to sequence file used as part of the configuration used with the Airbnb dataset.

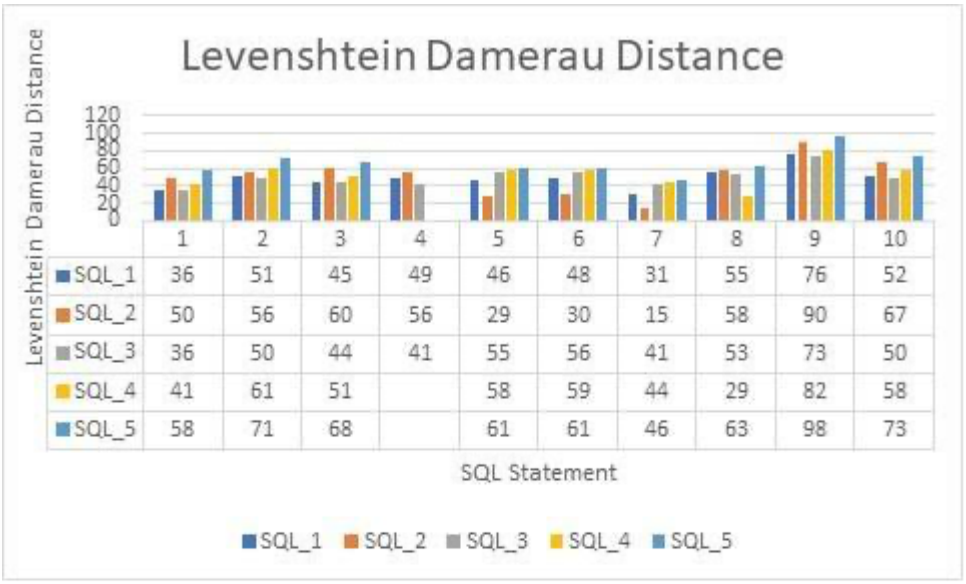
The last step of the natural language to SQL process is building the SQL statement. The model generates the statement *select neighbourhood_group,room_type from ab_nyc_2019 where neighbourhood_group like 'Brooklyn'* from the natural language query *looking for a room in Brooklyn*. Part of the reasoning behind using the Airbnb dataset was not just the size of the dataset but also all the data was stored in a single table. More work will need to be completed to show how the process would work with a query searching multiple tables.

6.4.1. SQL Validation

Having created the SQL statements the next step is to validate the accuracy of the conversion. The thesis is proposing to use two metrics to perform the marking process. The first of these metrics is the Jaro Winkler distance metric (1999) and the second is the Damerau Levenshtein distance. The work carried out by Cahyono (2019) and Zhao (2019) highlight a use case for using these algorithms. The advantage of using Damerau Levenshtein is the algorithm takes into consideration character insertion, deletion and transposition as described by Damerau (1964). The approach taken by Jaro Winkler (1999) uses a prefix scale p which gives a more favourable rating to strings that match from the beginning for a set prefix length l.

Both the Levenshtein Damerau and the Jaro Winkler distance measure the similarity between two strings. The two strings in question are firstly the SQL statement that the algorithm created by this chapter and the second is the SQL statement that is expected from the translation process. Pitchaimalai et al (2008) proposed using the Euclidean distance for evaluating the performance of SQL queries. The Levenshtein Damerau distance allows insertion, deletion, substitution, and the transposition of two adjacent characters. The Jaro Winkler metric compares the commonality between characters. The Euclidean distance treats the string as two vectors and compares the distance between them. It is arguable but when comparing SQL statements for this chapter the Euclidean distance did not prove to be as useful as Levenshtein Damerau and Jaro Winkler metrics. More research into this area is required as the current level of research appears to be lacking.

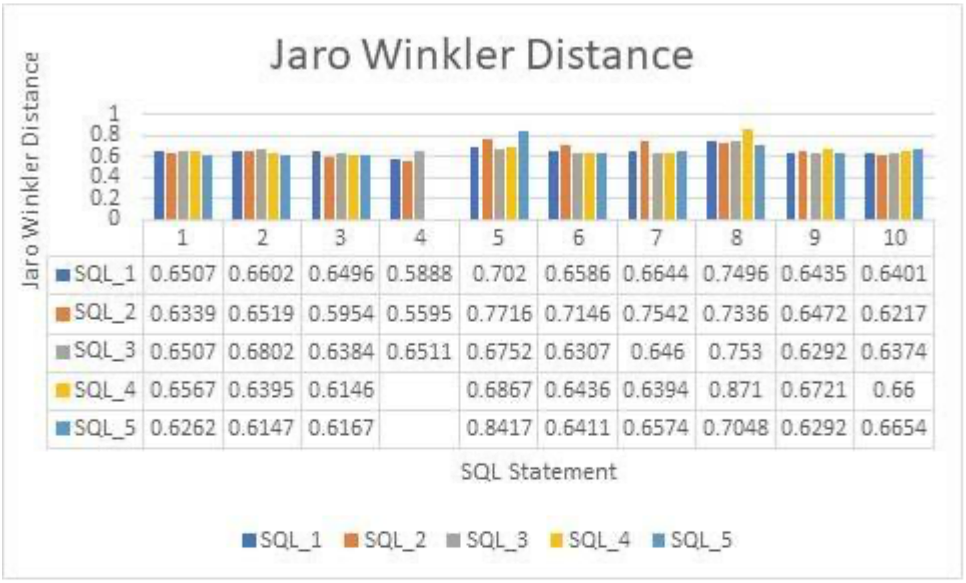
Appendix C shows a table extract of expected SQL statements compared with calculated statements. The calculated statements have been designed with the ‘UNION’ directive as it was found to be easier to compare two types of SQL statements using this clause. For comparison the compound calculated statement is separated into individual select statements separated by the ‘UNION’. The first five columns from both table 6.3 and 6.13 are for queries that have configuration within sequence-to-sequence configuration file. These queries have been tuned so keywords in the natural language input statement have been mapped to database objects such as table or column names. The remaining entries have no such configuration within the sequence to sequence model and as such can be classed as untuned.



	1	2	3	4	5	6	7	8	9	10
SQL_1	36	51	45	49	46	48	31	55	76	52
SQL_2	50	56	60	56	29	30	15	58	90	67
SQL_3	36	50	44	41	55	56	41	53	73	50
SQL_4	41	61	51		58	59	44	29	82	58
SQL_5	58	71	68		61	61	46	63	98	73

Table 6-.3: This table shows the results from measuring the SQL statements created by the algorithm proposed by this chapter and the expected output using the Levenshtein Damerau Distance. The underlying data for this table is shown in Appendix C.

From Table 6.3, each column represents an output from the algorithm which is made up of a number of SQL statements combined together using the SQL JOIN directive. Each row represents a SQL query separated by the JOIN directive. It should be noted that the resulting output from query 4 only has only three SQL statements as compared to the other queries having five SQL statements being joined.



	1	2	3	4	5	6	7	8	9	10
SQL_1	0.6507	0.6602	0.6496	0.5888	0.702	0.6586	0.6644	0.7496	0.6435	0.6401
SQL_2	0.6339	0.6519	0.5954	0.5595	0.7716	0.7146	0.7542	0.7336	0.6472	0.6217
SQL_3	0.6507	0.6802	0.6384	0.6511	0.6752	0.6307	0.646	0.753	0.6292	0.6374
SQL_4	0.6567	0.6395	0.6146		0.6867	0.6436	0.6394	0.871	0.6721	0.66
SQL_5	0.6262	0.6147	0.6167		0.8417	0.6411	0.6574	0.7048	0.6292	0.6654

Table 6-.4: This table shows the results from measuring the SQL statements created by the algorithm proposed by this chapter and the expected output using the Jaro Winkler Distance. The underlying data for this table is shown in Appendix C.

In Table 6.4 each column represents an output from the algorithm which is made up of a number of SQL statements combined together using the SQL JOIN directive. Each row represents a SQL query separated by the JOIN directive. It should be noted that the resulting output from query 4 only has only three SQL statements as compared to the other queries having five SQL statements being joined.

Using the Jaro Winkler distance the scores for the SQL statements range from 0.5 to 0.8 with an overall average of 0.6. The average score for the tuned SQL statements is 0.65 while the score for the untuned statements is 0.67. The closer the score is to 1 the more accurate the calculated output is to the expected output. Looking at the average scores sample test carried out within this chapter shows that the results are inconclusive but for query 5 the average score is a more respectable 0.7 and a maximum value of 0.84 is even closer to 1. The small sample from this chapter shows that the untuned data can be just as accurate as the tuned queries. With the Levenshtein Damerau Distance the closer the score is to 100 the more accurate the translation. The average score for the tuned SQL statements is 50, while the average score for the untuned statements is 56. Again while the score of 50 is inconclusive query 9 shows that an untrained query can obtain an average of 80.

The approach being proposed by this chapter does require further refinement and more data to test against. The results highlight the problem that marking an SQL statement, against an arbitrary idealised statement, could bias the results as the score is calculated by comparing two arbitrary strings. Further research will need to be carried out on comparing SQL produced by this chapter against an optimised SQL statement.

6.4.1.1. WikiSQL

Having tested the proposed approach with the Kaggle New York AirBnB dataset and using Levenshtein Damerau Distance as well as the Jaro Winkler Distance to measure the accuracy of the resulting SQL statements. The next step in validating the usefulness of the proposed approach is to test the model against the Victor Zhong et al (2017) WikiSQL dataset and show how it compares against other approaches. The WikiSQL corpus is a record of over 80,000 natural language questions that can be asked against a database, this corpus has been hand annotated, which this chapter uses for building up a grammar file. Along with the corpus the WikiSQL team provides the corresponding SQL statements and associated database tables for marking and testing any NLIDB solution.

Table 6.5 is an extract of the test dataset supplied as part of the WikiSQL project. It shows the natural language questions as well as some of the annotation that is supplied with the dataset. In testing the accuracy of the model proposed by this chapter the supplied annotation was used to build up the configuration required to run the model.

sel	Operator Index	Question	Condition	condsWords	Column Index	agg	table_id
2	0	What is terrence ross' nationality	Terrence Ross	NaN	0	0	1-10015132-16
5	0	What club was in toronto 1995-96	1995-96	NaN	4	0	1-10015132-16
5	0	which club was in toronto 2003-06	2003-06	NaN	4	0	1-10015132-16
5	0	how many schools or teams had jalen rose	Jalen Rose	NaN	0	3	1-10015132-16
2	0	Where was Assen held?	Assen	NaN	3	0	1-10083598-1

Table 6-.5: The table contains an extract of the WikiSQL dataset used to validate the model proposed by this chapter.

6.4.1.2. Tokenise

Using a sample input natural language query from the above data extract *'What is terrence ross' nationality'*. Along with the supplied annotation the first step is to tokenise the input string. From the annotation supplied by the dataset it is known that the string *Terrence Ross* is a conditional word or by using a name tagger the string can be identified as a name. Using the hand annotated WikiSQK corpus the extract the extract from the name tagger that identifies Terrence Ross as a player is

names=[(‘terrence ross’, ‘player’)]

6.4.1.3. Stop words

The next step is to remove the stop words from the string which will leave just the word *nationality*. The list of columns from the WikiSQL dataset table labelled as *‘1-10015132-16’* are *“Player”, “No”, “Nationality” “Position”, “Years in Toronto”, “School / Club Team”*. As can be seen from the column list there is a column called ‘nationality’. This then gives us a key word for the SQL statement

6.4.1.4. Sequence to Sequence

The last piece of configuration required to complete the transformation from natural language to SQL is the sequence to sequence file. From the original input statement of *'What is terrence ross nationality'* the only word left to deal with is *nationality*. It is already known that it is a column in the table so the next step is to configure this in the sequence to sequence file.

```
nationality 1-10015132-16, nationality

player      1-10015132-16, player

name        1-10015132-16, player
```

Figure 6-.3: Shows an extract of the sequence to sequence file used by this chapter. The data used to populate this table comes from the hand annotated WikiSQL dataset.

Using the sequence to sequence file shown in **figure 6.3** it is possible to identify that the word nationality is a column from the table *‘1-10015132-16’.*

6.4.1.5. Convert to SQL

This chapter proposes using a DSL to convert the JSON conditional construct [('terrence ross','player')] along with the output from the sequence to sequence model into the following SQL statement `select nationality from 1-10015132-16 where player = 'terrence ross'` . The flexibility provided by the DSL also enables the same conditional construct to be converted into other languages such as XQuery or in the case of WikiSQL into a format that could be marked by the python evaluate script.

6.4.1.6. Validation

When running the WikiSQL data through the system proposed by this chapter and using the WikiSQL utility to compare the results from the proposed process with the WikiSQL desired result, the WikiSQL marking gives a result of 92%. This can be compared to the best of the published results from the WikiSQL project which shows that Lyu (2020) scored 92.2% and He (2019) 91.8%. What is not discussed with these papers is their ability to generate queries that can be used against non relational databases. The use of an internal DSL which is proposed by this project can potentially add the flexibility to convert a natural language statement potential into a no sql query.

Model	Execution Accuracy	Exact Match Accuracy	Paper	Year
NL2SQL-RULE	89.2	83.7	Content Enhanced BERT-based Text-to-SQL Generation	2019
TypeSQl +TC	82.6		TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation	2018
Tranx	78.6	68.8	TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation	2018
STAMP+RL	74.6	61	Semantic Parsing with Syntax- and Table-Aware SQL Generation	2018
STAMP+RL	74.4	60.7	Semantic Parsing with Syntax- and Table-Aware SQL Generation	2018
TypeSQl +TC	73.5		TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation	2018
PT-MAML	68	62.8	Natural Language to Structured Query Generation via Meta-Learning	2018
Seq2SQL	59.4	48.3	Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning	2017
Seq2Seq	35.9	23.4	Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning	2017

Table 6.6: This table shows the top performing projects that have tested against the WikiSQL dataset (https://paperswithcode.com/sota/code-generation-on-wikisql)

The 92% that the WikiSQL marking system gave for the results generated for the Exact Match by this project can be compared to the published results of other projects listed in Table 6.6. Though the code used for this project is a proof of concept rather than a robust performance-oriented production ready application the results are favourable.

6.5. Conclusion

This chapter shows that a sequence-to-sequence model can translate a natural language statement into more than just another natural language such as Spanish. Most research is centred on translating one natural language to another, this chapter shows that sequence to sequence models can be used to translate a natural language statement into a computer language such as SQL. A SQL statement is after all another language, like natural languages it has syntax, grammar and vocabulary. The thesis also highlights how sequence to sequence models can be used to improve the overall accuracy as part of the conversion.

The use of Levenshtein Damerau and Jaro Winkler metrics to measure the accuracy of the conversion from natural language to SQL give some metric of success but the accuracy usefulness of such metrics needs to be examined further. For the purpose of this chapter using Jaro Winkler the results range from 55% to over 80% accuracy of all the conversions and Levenshtein Damerau marks the SQL conversion in an even greater range with values from 30% to over 90%.

The approach proposed in this chapter has been tested against two large datasets firstly against the Kaggle dataset and then against the dataset supplied by the WikiSQL project. Comparing the resulting SQL statement against an expected statement can be subjective. The results have been encouraging as the test datasets are unrelated yet the accuracy of the conversation for the WikiSQL data was 92% by their own marking. Though more work needs to be carried out the results show some potential for improving the accuracy of converting natural language to SQL. Having defined the core as a potential solution to the problem of converting a natural language statement into a language capable of querying a database, the next step is to provide an interface using the approach proposed by this research. For that the project is proposing to design an overarching external DSL.

7. Conclusion

The idea behind this project was to provide the viability and the performance improvements to convert a natural language to a structured query language capable of querying a database. As has been discussed by this project the current solutions lack a commercial viability which manifests itself in the lack of solutions providing an application that can extract data from a repository based on a natural language query.

Part of this problem has been highlighted by several researchers which shows the performance of such systems are lacking in both speed and accuracy. During the research phase of this project, it became apparent that there was also a gap in how to implement the solutions that research teams were proposing. Research projects were refining the algorithms used to convert natural language into another language capable of searching a repository. However, there was little to no consideration on implementing the refined algorithms. Researchers such as Polosukhin as well as Skeggs & Lauria has briefly highlighted the issue but had not proposed a solution. As a result this thesis proposes an algorithm using shallow parsing that could be used to convert a natural language statement into a language capable of extracting data from a database repository. The second aim was to provide a common interface to enable retrieval to take place. To this extent the project has proposed creating a domain specific language (DSL) that can be used as an interface into the natural language shallow parsing algorithm.

7.1. Contribution

The original contribution being proposed by this project is made up of two elements. The first is a new algorithm based on a shallow parsing of the natural language statement as part of the conversion to a language capable of querying a repository. The aim being to improve the time to conversion and the accuracy of the SQL output from the conversion process. The second element is the development of a domain specific language (DSL) that can reduce the need for relying on language nuance and help with the conversion process.

7.1.1. Shallow Parsing

The first element of originality being proposed by this project is the shallow parsed algorithm from chapter 5. The algorithm is unique in a number of ways, first it proposes to combine a number of differing techniques to convert natural language into SQL. The project also demonstrates the advantage in speed of performance of the conversion process against similar solutions.

In Chapter 6, the project goes even further in the use of shallow parsing to refine the process converting natural language to a structured query language. Through the combination of shallow parsing and the use of RNN within sequence to sequence modelling the conversion process can be improved in terms of both accuracy and speed when compared to existing approaches. Most research projects had concentrated on refining a single approach to converting natural language to SQL. This project proposes combining techniques as well as proposing to use domain specific languages to solve the interface problem as well as assisting within the conversion process.

The performance enhancements of the approach proposed by this project have been addressed in both chapters 5 and 6. Chapter 5 highlighted the improvements in speed when using shallow parsing to perform the conversion from a natural language to a language capable of querying a data repository.

There are no performance metrics when it comes to the duration of the conversion process. Table 7.1 provides one of the few speed-based metrics when it comes to measuring the time a conversion process takes. Because there is no direct comparison with the datasets from Table 7.1 speed comparisons may be regarded as being speculative.

Type of Data	**No of words**	**Time Required by QTAG (Used in Enlight)**	**Time Required by Minipar (Used in Sapere)**
Times of India	202	1.71 secs	2.88 secs
Reply START QASystem (251Words) University Information	251	3.11 secs	
NMU Broadcaster	226	1.55 secs	2.86 secs
Wikipedia	226	1.67 secs	3.13 secs
Average		1.705 secs	2.9925 secs

Table 7-.1: is actually table 5.5 and is used to highlight the speed of conversion when using the approach being proposed by the project.

However, when looking at the speed comparisons between the datasets being referenced in Table 7.1 and the dataset used in Chapter 5 for converting natural language to a languga capable of querying a database (table 7.2). It should be noted that the dataset referenced in table 7.2 contains more data than those referenced in table 7.1. The speed of conversion and extraction is faster using the approach being proposed by this project in Chapter 5 even though the data used in this test has far more data than the datasets in table 7.1.

Collection	Number of Words	SQL Conversion	Data Extraction
ginf.csv	19531	0.665 secs	0.9 secs

Table 7-.2: is actually table 5.6. Performance from the proposed system which includes the conversion from natural language to SQL..

The same can also be true when it comes to the accuracy of the conversion process being proposed by this thesis. The details of the accuracy can be seen in section 6.4. The use of Levenshtein Damerau and Jaro Winkler metrics to measure the accuracy of the conversion from natural language to SQL give some metric of success but the accuracy usefulness of such metrics needs to be examined further. For the purpose of this project using Jaro Winkler the results range from 55% to over 80% accuracy of all the conversions and Levenshtein Damerau marks the SQL conversion in an even greater range with values from 30% to over 90%.

The approach proposed in this project has been tested against two large datasets firstly against the Kaggle dataset and then against the dataset supplied by the WikiSQL project. Comparing the resulting SQL statement against an expected statement can be subjective. The results have been encouraging as the test datasets are unrelated yet the accuracy of the conversation for the WikiSQL data was 92% by their own marking. The results of the marking process can then be compared to the published results from the WikiSQL project which are shown in Table 6.6. The 92% compares against the next highest mark of 89%. Though more work needs to be carried out the results show some potential for improving the accuracy of converting natural language to SQL. Having defined that shallow parsing is a potential solution to the problem of converting a natural language statement into a language capable of querying a database, the next step is to provide an interface using the approach proposed by this research. For that the project is proposed the design of an overarching external DSL.

7.1.2. Domain Specific Language

The second contribution this project adds to the field of natural language interface to a database is the introduction of a domain specific language (DSL). Creating and using a DSL might sound counterintuitive to the NLIDB problem. Using Bloom's knowledge taxonomy as the construct of the DSL still allows for the proposal in this project to be classed as an NLIDB solution. The DSL also provides a common interface into using algorithms to convert natural language to SQL. Initially the Xtext application was proposed to create a framework for building the DSL. Xtext proved to be useful for creating a quick mock-up of a suitable grammar file for validating a proof of concept for the language. However, with more testing and expanding the grammar of the DSL using the Xtext grammar files the process began to become difficult to modify and extend the DSL grammar file. The final idea was to use Xtext to just validate the structure of the input query based on the Bloom taxonomy.

There are two advantages to using the DSL and these advantages are:

1. The advantage of using a DSL provides for a simplified process. The grammar file used to parse the incoming natural language statement can be simplified as the structure of the DSL is known. With more development it may even be possible to rewrite the entire translation process, streamlining how the conversion is performed.
2. Currently within this project XText is only being used to validate the incoming query. The algorithms used and discussed within chapters 5 & 6 could also be refined to work more closely with the DSL. This tighter binding between the parsing algorithm and the DSL would negate the need for XText.

7.2. Limitations

The full implementation of this project has only been tested against sample datasets, which is designed to show that the solution proposed by this project works. A full implementation within a commercial setting has yet to be completed. During testing the project it became apparent that there is the potential for some limitations on the system when deployed into a commercial environment. These limitations are listed below.

- The grammar file used by Xtext for creating a grammar from chapter 4 is used for defining the DSL and has the potential to become unwieldy. The size of this file may make extending the grammar rules for the DSL too complicated and prone to errors. The size of the file may also ensure that there are size requirements for storing such a large file. The size and complexity of the grammar file may also have a negative impact on the performance of the DSL parse in terms of speed and accuracy.

- The grammar files used by the OpenNLP described in chapter 5 library also have the potential to keep the configuration up to date cumbersome and prone to error. These files could also have an impact on storage requirements as well as the speed of conversion if they became too large.

- The sequence-to-sequence model described in chapter 6 uses text files to define how words can be translated from one domain into another. These text files again have the potential to become exceedingly large and difficult to manage. Along with the difficulty this could potentially pose with maintaining the files to ensure the content is kept up to date. The size could also have a negative impact on the speed of conversion as well as the potential for special storage requirements.

More testing is required against large and more complex environments to fully understand the limitations of the solution being proposed by this thesis.

7.3. Future Work

The algorithms used to create both the domain specific language and the shallow passed algorithm have been written as proof of concept code. The code will need to be enhanced and parts rewritten to make the code production ready. When the code rewrite is complete it will become available via a Git repository. It must also be added that currently the framework proposed by this project has been implemented in a number of various guises in four projects that I have worked on. A complete implementation of all the points proposed by this project has yet to be implemented.

There is also an argument for pursuing ISO accreditation for the domain specific language that would then lead to a wider audience. Before ISO accreditation can be achieved a tighter definition and promotion of the language grammar syntax would need to be completed. Future refinements of the DSL are also actively encouraged and a platform supporting the development and refinement of the language will need to be developed.

Currently this project concentrates on converting natural language statements to SQL. Chapter 4 and 6 introduce the concept of an internal DSL which can be used as part of the conversion process. The internal DSL has not been implemented within this thesis, but the concept of the internal DSL can be used to convert the natural language to a query language other than SQL.

Appendix A – NLP Tags

The list of NLP tags that are used by this project to determine which how to deal with the input word tokens. The tags are based the tags that are used within the OpenNLP project.

Parts of Speech	Meaning of parts of speech
NN	Noun, singular or mass
DT	Determiner
VB	Verb, base form
VBD	Verb, past tense
VBZ	Verb, third person singular present
IN	Preposition or subordinating conjunction
NNP	Proper noun, singular
TO	to
JJ	Adjective
IRR	Irrelevant and can be ignored.

Appendix B – AirBnB metadata

The table describes the meta data of the Airbnb dataset used in Chapter 6.

Variable Name	**Data Type**
Id	Numeric
Name	String
Host_id	Numeric
Host_name	Sting
neighbourhood_group	String
neighbourhood	String
latitude	Numeric
longitude	Numeric
room_type	String
price	Numeric
minimum_nights	Numeric
number_of_reviews	Numeric
last_review	Date
reviews_per_month	numeric
calculated_host_listings_count	Numeric
availability_365	Numeric

Appendix C - WikiSQL

The table below shows the output from the algorithm used in Chapter 6, when the approach is used against the WikiSQL dataset. This table shows the algorithm generated SQL statements against the expect SQL as devised by the WikiSQL project along withe the markings calculated using the Damerau and the Jaro algorithms. The data in the table is the source of data for the tables 6.12 and 6.13.

NLP Question: The WikiSQL input statement

Calculated SQL: The SQL output from the proposed model.

Expected SQL: The SQL expected from the output.

Damerau: The Demerau similarity between the expected output and the Calculated output. The Calculated output uses multiple queries and uses union to join them. The marking takes the individual statement and marks them. The results are the individual scores.

Jaro: The Jaro similarity between the expected output and the Calculated output. The Calculated output uses multiple queries and uses union to join them. The marking takes the individual statement and marks them. The results are the individual scores.

NLP Question	Calculated SQL	Expected SQL	Damerau	Jaro
What is the local name given to the city of Canberra?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'city'= 'Canberra' union	select org_name from organisations where ORGANIZATION like 'Canberra'	36	0.65069347
	select 'city', 'country', 'org_name' from organisations where 'city'= 'Canberra' union		50	0.633942221
	select 'lastname' from contacts where or 'city'= 'Canberra'		36	0.65069347
	select 'school_name', 'club_name' from sports_clubs where 'city'= 'Canberra' union		41	0.656749381
	select 'team_name' from sports_club where 'city'= 'Canberra'		58	0.62615075
Which teams won when Bobby Rahal was their winning driver?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'person'= 'Bobby' or 'person'= 'Rahal' union	select contacts, lastname from person where lastname like Rahal	51	0.660164879
	select 'city', 'country', 'org_name' from organisations where 'person'= 'Bobby' or 'person'= 'Rahal' union		56	0.651948052
	select 'lastname' from contacts where 'person'= 'Bobby' or 'person'= 'Rahal' union		50	0.680152601
	select 'school_name', 'club_name' from sports_clubs where 'person'= 'Bobby' or 'person'= 'Rahal' union		61	0.639483913
	select 'team_name' from sports_club where 'person'= 'Bobby' or 'person'= 'Rahal'		71	0.61468254
What school or club team is Amir Johnson on?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'person'= 'Amir' or 'person'= 'Johnson' union	select org_name from organisation where ORGANIZATION like 'Johnson'	45	0.649559432
	select 'city', 'country', 'org_name' from organisations where 'person'= 'Amir' or 'person'= 'Johnson' union		60	0.595422886
	select 'lastname' from contacts where 'person'= 'Amir' or 'person'= 'Johnson' union		44	0.638423952
	select 'school_name', 'club_name' from sports_clubs where 'person'= 'Amir' or 'person'= 'Johnson' union		51	0.614573146
	select 'team_name' from sports_club where 'person'= 'Amir' or 'person'= 'Johnson'		68	0.616723081
looking for a room in Brooklyn	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'city'= 'Brooklyn' union	select AB_NYC_2019, room_type from ab_nyc_2019 where ORGANIZATION like 'Brooklyn'	49	0.588821991
	select 'city', 'country', 'org_name' from organisations where 'city'= 'Brooklyn' union		56	0.559475622
	select 'lastname' from contacts where 'city'= 'Brooklyn'		41	0.651079769

What is the team located at philips arena 18?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'ORGANIZATION'= 'Philips' or 'ORGANIZATION'= 'Arena' union	select 'school_name', 'club_name' from sports_clubs where 'ORGANIZATION '= 'Philips'	46	0.70198273
	select 'city', 'country', 'org_name' from organisations where 'ORGANIZATION'= 'Philips' or 'ORGANIZATION'= 'Arena' union		29	0.771597134
	select 'lastname' from contacts where 'ORGANIZATION'= 'Philips' or 'ORGANIZATION'= 'Arena' union		55	0.675170614
	select 'school_name', 'club_name' from sports_clubs where 'ORGANIZATION'= 'Philips' or 'ORGANIZATION'= 'Arena' union		58	0.686731787
	select 'team_name' from sports_club where 'ORGANIZATION'= 'Philips' or 'ORGANIZATION'= 'Arena'		61	0.841704079

What season features writer Michael Poryes?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'ORGANIZATION'= 'Michael' or 'ORGANIZATION'= 'Poryes' union	select 'school_name', 'club_name' from sports_clubs where 'lastname'= 'Michael' or lastname='Pory es' "	48	0.658642724
	select 'city', 'country', 'org_name' from organisations where 'ORGANIZATION'= 'Michael' or 'ORGANIZATION'= 'Poryes' union		30	0.714551084
	select 'lastname' from contacts where 'ORGANIZATION'= 'Michael' or 'ORGANIZATION'= 'Poryes' union		56	0.630672254
	select 'school_name', 'club_name' from sports_clubs where 'ORGANIZATION'= 'Michael' or 'ORGANIZATION'= 'Poryes' union		59	0.643629113
	select 'team_name' from sports_club where 'ORGANIZATION'= 'Michael' or 'ORGANIZATION'= 'Poryes'		61	0.641079315

Which visitors have a leading scorer of Roy	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'ORGANIZATION'= 'Roy' union	select 'school_name', 'club_name' from sports_clubs where 'lastname'= 'Roy'	31	0.664444444
	select 'city', 'country', 'org_name' from organisations where 'ORGANIZATION'= 'Roy' union		15	0.754215103
	select 'lastname' from contacts where 'ORGANIZATION'= 'Roy' union		41	0.645974224
	select 'school_name', 'club_name' from sports_clubs where 'ORGANIZATION'= 'Roy' union		44	0.639440921
	select 'team_name' from sports_club where 'ORGANIZATION'= 'Roy'		46	0.657438672

What is the location of the Carousel toll plaza?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Carousel' union	"select 'city', 'country', 'org_name' from organisations where 'ORGANIZATION'= 'Carousel'"	55	0.749606258
	select 'city', 'country', 'org_name' from organisations where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Carousel' union		58	0.733595608
	select 'lastname' from contacts where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Carousel' union		53	0.752954035
	select 'school_name', 'club_name' from sports_clubs where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Carousel' union		29	0.871049924
	select 'team_name' from sports_club where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Carousel'		63	0.704758632
What is the rank of manager Rob Mcdonald?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Rob' or 'ORGANIZATION'= 'Mcdonald' union	Select rank from organisations where person = 'Rob' or person = 'Mcdonald'	76	0.643498242
	select 'city', 'country', 'org_name' from organisations where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Rob' or 'ORGANIZATION'= 'Mcdonald' union		90	0.647154778
	select 'lastname' from contacts where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Rob' or 'ORGANIZATION'= 'Mcdonald' union		73	0.629231785
	select 'school_name', 'club_name' from sports_clubs where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Rob' or 'ORGANIZATION'= 'Mcdonald' union		82	0.672111457
	select 'team_name' from sports_club where 'ORGANIZATION'= 'What' or 'ORGANIZATION'= 'Rob' or 'ORGANIZATION'= 'Mcdonald'		98	0.629215485
What were the results for incumbent Jim McCrery?	select 'neighbourhood_group', 'room_type' from ab_nyc_2019 where 'ORGANIZATION'= 'Jim' or 'ORGANIZATION'= 'McCrery' union	select votes from organisations where person = 'Jim or person = 'McCrery'	52	0.640116407
	select 'city', 'country', 'org_name' from organisations where 'ORGANIZATION'= 'Jim' or 'ORGANIZATION'= 'McCrery' union		67	0.621724813
	select 'lastname' from contacts where 'ORGANIZATION'= 'Jim' or 'ORGANIZATION'= 'McCrery' union		50	0.637406188

	select 'school_name', 'club_name' from sports_clubs where 'ORGANIZATION'= 'Jim' or 'ORGANIZATION'= 'McCrery' union		58	0.660036356
	select 'team_name' from sports_club where 'ORGANIZATION'= 'Jim' or 'ORGANIZATION'= 'McCrery'		73	0.665442741

Bibliography

'X-SQL: reinforce schema representation with context', (2019) .

MPS: The Domain-Specific Language Creator by JetBrains. Available at: https://www.jetbrains.com/mps/ (Accessed: .

Xtext - Language Engineering Made Easy! Available at: https://www.eclipse.org/Xtext/ (Accessed: .

14:00-17:00 *ISO/IEC CD 9075-2.* Available at: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/65/76584.html (Accessed: .

Agarwal, R., Liang, C., Schuurmans, D. and Norouzi, M. (2019) 'Learning to Generalize from Sparse and Underspecified Rewards', *arXiv:1902.07198 cs, stat],* .

Ahkouk, K., Machkour, M., Ennaji, M., Erraha, B. and Antari, J. (2019) *Comparative study of existing approaches on the Task of Natural Language to Database Language.* . 07. pp. 1.

Allamanis, M., Barr, E.T., Devanbu, P. and Sutton, C. (2019) 'A Survey of Machine Learning for Big Code and Naturalness', *ACM Computing Surveys,* 51(4), pp. 1-37. doi: 10.1145/3212695.

Anderson, L.W., Krathwohl, D.r. and Bloom, B.S. (2001) *A taxonomy for learning, teaching, and assessing : a revision of Bloom's taxonomy of educational objectives /.* Longman.

Androutsopoulos, I., Ritchie, G.D. and Thanisch, P. (1995) 'Natural language interfaces to databases – an introduction', *Natural Language Engineering,* 1(1), pp. 29-81. doi: 10.1017/S135132490000005X.

Azuma, M., Coallier, F. and Garbajosa, J. (2003) *How to apply the Bloom taxonomy to software engineering.* . 09. pp. 117.

Badhya, S.S., Prasad, A., Rohan, S., Yashwanth, Y.S., Deepamala, N. and Shobha, G. (2019) *Natural Language to Structured Query Language using Elasticsearch for descriptive columns.* . 12. pp. 1.

Baik, C., Jagadish, H.V. and Li, Y. (2019) 'Bridging the Semantic Gap with SQL Query Logs in Natural Language Interfaces to Databases', *2019 IEEE 35th International Conference on Data Engineering (ICDE),* , pp. 374-385. doi: 10.1109/ICDE.2019.00041.

Barišić, A., Amaral, V. and Goulão, M. (2018) 'Usability driven DSL development with USE-ME', *Computer Languages, Systems & Structures,* 51, pp. 118-157. doi: 10.1016/j.cl.2017.06.005.

Barišić, A., Blouin, D., Amaral, V. and Goulão, M. (2017) *A requirements engineering approach for usability-driven DSL development.* . 10/23; 2021/01/06. Association for Computing Machinery, pp. 115.

Barringer, H. and Havelund, K. (2012) *Internal versus External DSLs for Trace Analysis.* Springer Berlin Heidelberg, pp. 1.

Béchet, F., Nasr, A. and Genet, F. (2000) *Tagging Unknown Proper Names Using Decision Trees.* . 10; 2021/09/12. Association for Computational Linguistics, pp. 77.

Bello, I., Kulkarni, S., Jain, S., Boutilier, C., Chi, E., Eban, E., Luo, X., Mackey, A. and Meshi, O. (2019) 'Seq2Slate: Re-ranking and Slate Optimization with RNNs', *arXiv:1810.02019 cs, stat],* .

Benajiba, Y., Sun, J., Zhang, Y., Jiang, L., Weng, Z. and Biran, O. (2018) 'Siamese Networks for Semantic Pattern Similarity', *arXiv:1812.06604 cs],* .

Bentley, J. (1986) 'Programming pearls: little languages', *Communications of the ACM,* 29(8), pp. 711-721. doi: 10.1145/6424.315691.

Bird, S., Klein, E., Loper, E. and Baldridge, J. (2008) *Multidisciplinary instruction with the Natural Language Toolkit.* . 06/19; 2021/09/12. Association for Computational Linguistics, pp. 62.

Bogin, B., Gardner, M. and Berant, J. (2019) 'Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing', *arXiv:1905.06241 cs],* .

Boronat, A. 'Expressive and Efficient Model Transformation with an Internal DSL of Xtend', .

Brad, F., Iacob, R., Hosu, I. and Rebedea, T. (2017) 'Dataset for a Neural Natural Language Interface for Databases (NNLIDB)', *arXiv:1707.03172 cs],* .

Brad, F., Iacob, R., Hosu, I., Ruseti, S. and Rebedea, T. (2018) *A Syntax-Guided Neural Model for Natural Language Interfaces to Databases.* . 11. pp. 229.

Bunder, H. 'A UML-Agnostic Migration Approach From UML to DSL', , pp. 2.

Cahyono, S.C. (2019) 'Comparison of document similarity measurements in scientific writing using Jaro-Winkler Distance method and Paragraph Vector method', *IOP Conference Series: Materials Science and Engineering,* 662(5), pp. 052016. doi: 10.1088/1757-899X/662/5/052016.

Campagna, G., Xu, S., Moradshahi, M., Socher, R. and Lam, M.S. (2019) *Genie: a generator of natural language semantic parsers for virtual assistant commands.* . 06/08; 2022/02/15. Association for Computing Machinery, pp. 394.

Chen, X., Liu, C. and Song, D. (2019) 'EXECUTION-GUIDED NEURAL PROGRAM SYNTHESIS', , pp. 15.

Cheng, J., Reddy, S. and Lapata, M. (2018) 'Building a Neural Semantic Parser from a Domain Ontology', *arXiv:1812.10037 cs],* .

Cheng, J., Reddy, S., Saraswat, V. and Lapata, M. (2019) 'Learning an Executable Neural Semantic Parser', *Computational Linguistics,* 45(1), pp. 59-94. doi: 10.1162/coli_a_00342.

Chiu, C., Sainath, T.N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R.J., Rao, K., Gonina, E., Jaitly, N., Li, B., Chorowski, J. and Bacchiani, M. (2018) *State-of-the-Art Speech Recognition with Sequence-to-Sequence Models.* . 04. pp. 4774.

Cho, M., Amplayo, R.K., Hwang, S. and Park, J. (2018) 'Adversarial TableQA: Attention Supervision for Question Answering on Tables', *arXiv:1810.08113 cs],* .

Choi, D., Shin, M.C., Kim, E. and Shin, D.R. (2020) 'RYANSQL: Recursively Applying Sketch-based Slot Fillings for Complex Text-to-SQL in Cross-Domain Databases', *arXiv:2004.03125 cs],* .

Chuan, L., Ozcan, F., Quamar, A., Mittal, A., Sen, J., Saha, D. and Sankaranarayanan, K. (2018) 'Ontology-Based Natural Language Query Interfaces for Data Exploration', 41(3).

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014) 'Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling', *arXiv:1412.3555 cs],* .

Cleenewerck, T. (2003) *Component-based DSL Development.* Springer-Verlag, pp. 245.

Cuadrado, J.S., Izquierdo, J.L.C. and Molina, J.G. (2013) *Comparison Between Internal and External DSLs via RubyTL and Gra2MoL.* Available at: (Accessed: .

Czarnecki, K., O'Donnell, J., Striegnitz, J.ö and Taha, W. (2003) *DSL implementation in MetaOCaml, template Haskell, and C++.*

Dadashkarimi, J. and Tatikonda, S. (2018) 'Sequence to Logic with Copy and Cache', *arXiv:1807.07333 cs, stat],* .

Damerau, F.J. (1964) 'A technique for computer detection and correction of spelling errors', *Communications of the ACM,* 7(3), pp. 171-176. doi: 10.1145/363958.363994.

Desai, A., Gulwani, S., Hingorani, V., Jain, N., Karkare, A., Marron, M., R, S. and Roy, S. (2016) *Program synthesis using natural language.* . 05/14; 2020/12/30. ACM, pp. 345.

Deuter, A. and Koch, H. (2015) *Applying Manufacturing Performance Figures to Measure Software Development Excellence.* Springer International Publishing, pp. 62.

Ellis, K., Ritchie, D., Solar-Lezama, A. and Tenenbaum, J. (2018) *Learning to Infer Graphics Programs from Hand-Drawn Images.* . 2022/02/15. Curran Associates, Inc, .

Finegan-Dollak, C., Kummerfeld, J.K., Zhang, L., Ramanathan, K., Sadasivam, S., Zhang, R. and Radev, D. (2018) 'Improving Text-to-SQL Evaluation Methodology', *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* , pp. 351-360. doi: 10.18653/v1/P18-1033.

Gallant, S.I. (1990) 'Perceptron-based learning algorithms', *IEEE Transactions on Neural Networks,* 1(2), pp. 179-191. doi: 10.1109/72.80230.

Gama, K., Pedraza, G., Lévêque, T. and Donsez, D. (2011) *Application management plug-ins through dynamically pluggable probes.* . 05/28; 2021/04/28. Association for Computing Machinery, pp. 32.

Gómez-Rodríguez, C., Alonso-Alonso, I. and Vilares, D. (2019) 'How important is syntactic parsing accuracy? An empirical evaluation on rule-based sentiment analysis', *Artificial Intelligence Review,* 52(3), pp. 2081-2097. doi: 10.1007/s10462-017-9584-0.

Gulcu, A. (2016) 'The Evaluation of the Cognitive Learning Process of the Renewed Bloom Taxonomy Using a Web Based Expert System', *The Turkish Online Journal of Educational Technology,* 15(4), pp. 17.

Gulwani, S. and Marron, M. (2014) *NLyze: interactive programming by natural language for spreadsheet data analysis and manipulation.* . 06/18; 2020/12/30. Association for Computing Machinery, pp. 803.

Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J., Liu, T. and Zhang, D. (2019) 'Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation', *arXiv:1905.08205 cs],* .

Guo, T. and Gao, H. (2018) 'Bidirectional Attention for SQL Generation', *arXiv:1801.00076 cs],* .

Gur, I., Yavuz, S., Su, Y. and Yan, X. (2018) *DialSQL: Dialogue Based Structured Query Generation.* . 07; 2022/02/15. Association for Computational Linguistics, pp. 1339.

Han, X., Hu, L., Sen, J., Dang, Y., Gao, B., Isahagian, V., Lei, C., Efthymiou, V., Özcan, F., Quamar, A., Huang, Z. and Muthusamy, V. (2020) *Bootstrapping Natural Language Querying on Process Automation Data.* . 11. pp. 170.

Hanane, B., Machkour, M. and Koutti, L. (2020) 'A model of a generic Arabic language interface for multimodel database', *International Journal of Speech Technology,* 23(3), pp. 669-681. doi: 10.1007/s10772-020-09740-9.

Heeren, B. and Jeuring, J. (2017) *An Extensible Domain-Specific Language for Describing Problem-Solving Procedures.* Springer International Publishing, pp. 77.

Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D. and Slocum, J. (1978) 'Developing a natural language interface to complex data', *ACM Transactions on Database Systems,* 3(2), pp. 105-147. doi: 10.1145/320251.320253.

Hinkel, G., Goldschmidt, T., Burger, E. and Reussner, R. (2019) 'Using internal domain-specific languages to inherit tool support and modularity for model transformations', *Software & Systems Modeling,* 18(1), pp. 129-155. doi: 10.1007/s10270-017-0578-9.

Hinkel, G. and Happe, L. 'Using component frameworks for model transformations by an internal DSL', . doi: 10.13140/2.1.4629.7289.

Huang, P., Wang, C., Singh, R., Yih, W. and He, X. (2018) 'Natural Language to Structured Query Generation via Meta-Learning', *arXiv:1803.02400 cs],* .

Iacob, R.C.A., Brad, F., Apostol, E., Truică, C., Hosu, I.A. and Rebedea, T. (2020) *Neural Approaches for Natural Language Interfaces to Databases: A Survey.* . 12; 2020/12/30. International Committee on Computational Linguistics, pp. 381.

Jia, R. and Liang, P. (2016) *Data Recombination for Neural Semantic Parsing.* . 08; 2022/02/15. Association for Computational Linguistics, pp. 12.

Joshi, S.R., Venkatesh, B., Thomas, D., Jiao, Y. and Roy, S. (2020) *A Natural Language and Interactive End-to-End Querying and Reporting System.* . 01/05; 2022/02/15. Association for Computing Machinery, pp. 261.

Joshi and Akerkar (2008) 'ALGORITHMS TO IMPROVE PERFORMANCE OF NATURAL LANGUAGE INTERFACE', *International Journal of Computer Science & Applications,* 5(2), pp. 52-68.

JoshuaPartlow *Getting Started with Domain-Specific Languages - Visual Studio.* Available at: https://docs.microsoft.com/en-us/visualstudio/modeling/getting-started-with-domain-specific-languages (Accessed: .

Jwalapuram, P. and Mamidi, R. (2017) *Domain independent keyword identification for question answering.* . 12. pp. 95.

Kalajdjieski, J., Toshevska, M. and Stojanovska, F. (2020) 'Recent Advances in SQL Query Generation: A Survey', *arXiv:2005.07667 cs],* .

Kamath, A. and Das, R. (2019) 'A Survey on Semantic Parsing', *arXiv:1812.00978 cs],* .

Kapferer, S. (2019) *Model Transformations for DSL Processing.* Available at: https://eprints.hsr.ch/819/ (Accessed: .

Karki, B., Hu, F., Haridas, N., Barot, S., Liu, Z., Callebert, L., Grabmair, M. and Tomasic, A. (2019) *Question answering via web extracted tables.* . 07/05; 2022/02/15. Association for Computing Machinery, pp. 1.

Keneshloo, Y., Shi, T., Ramakrishnan, N. and Reddy, C.K. (2019) 'Deep Reinforcement Learning For Sequence to Sequence Models', *arXiv:1805.09461 cs, stat],* .

Kim, M. and Kim, H. (2018) *Dialogue Act Classification Model Based on Deep Neural Networks for a Natural Language Interface to Databases in Korean.* . 01. pp. 537.

Kingma, D.P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980 cs],* .

Klint, P., van der Storm, T. and Vinju, J. (2009) *RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation.* . 09; 2020/12/27. IEEE, pp. 168.

Koutti, L., Machkour, M. and Bais, H. (2018) 'An Arabic natural language interface for querying relational databases based on natural language processing and graph theory methods', *International Journal of Reasoning-based Intelligent Systems,* 10, pp. 155. doi: 10.1504/IJRIS.2018.10013299.

Křikava, F. and Collet, P. (2012) *On the use of an internal DSL for enriching EMF models.* . 09/30; 2020/12/27. Association for Computing Machinery, pp. 25.

Landhäußer, M., Weigelt, S. and Tichy, W.F. (2017) 'NLCI: a natural language command interpreter', *Automated Software Engineering,* 24(4), pp. 839-861. doi: 10.1007/s10515-016-0202-1.

Langlois, B., Jitia, C. and Jouenne, E. 'DSL Classification', , pp. 11.

Li, X. and Roth, D. (2001) *Exploring evidence for shallow parsing.* . 2021/03/07.

Li, Y., Feng, A., Li, J., Mumick, S., Halevy, A., Li, V. and Tan, W. (2019) 'Subjective databases', *Proceedings of the VLDB Endowment,* 12(11), pp. 1330-1343. doi: 10.14778/3342263.3342271.

Li, Y. and Rafiei, D. (2017) *Natural Language Data Management and Interfaces: Recent Development and Open Challenges.* . 05/09; 2021/09/12. Association for Computing Machinery, pp. 1765.

Liang, C., Norouzi, M., Berant, J., Le, Q.V. and Lao, N. (2018) *Memory Augmented Policy Optimization for Program Synthesis and Semantic Parsing.* . 2022/02/15. Curran Associates, Inc, .

Lin, K., Bogin, B., Neumann, M., Berant, J. and Gardner, M. (2019) 'Grammar-based Neural Text-to-SQL Generation', *arXiv:1905.13326 cs],* .

Lin, X.V., Wang, C., Zettlemoyer, L. and Ernst, M.D. (2018a) 'NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System', *arXiv:1802.08979 cs],* .

Lin, X.V., Wang, C., Zettlemoyer, L. and Ernst, M.D. (2018b) 'NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System', *arXiv:1802.08979 cs],* .

Liu, S., Cardenas, A., Xiong, X., Mernik, M., Bryant, B. and Gray, J. (2010) 'A SOA Approach for Domain-Specific Language Implementation', *2010 6th World Congress on Services,* . doi: 10.1109/SERVICES.2010.119.

Llopis, M. and Ferrández, A. (2013) 'How to make a natural language interface to query databases accessible to everyone: An example', *Computer Standards & Interfaces,* 35(5), pp. 470-481. doi: 10.1016/j.csi.2012.09.005.

Lukovnikov, D., Chakraborty, N., Lehmann, J. and Fischer, A. (2018) 'Translating Natural Language to SQL using Pointer-Generator Networks and How Decoding Order Matters', *arXiv:1811.05303 cs],* .

Lyu, Q., Chakrabarti, K., Hathi, S., Kundu, S., Zhang, J. and Chen, Z. (2020) 'Hybrid Ranking Network for Text-to-SQL', .

McCann, B., Keskar, N.S., Xiong, C. and Socher, R. (2018) 'The Natural Language Decathlon: Multitask Learning as Question Answering', *arXiv:1806.08730 cs, stat],* .

Méndez-Acuña, D., Galindo, J.A., Degueule, T., Combemale, B. and Baudry, B. (2016) 'Leveraging Software Product Lines Engineering in the development of external DSLs: A systematic literature review', *Computer Languages, Systems & Structures,* 46, pp. 206-235. doi: 10.1016/j.cl.2016.09.004.

Mengerink, J.G.M., van der Sanden, B., Cappers, B.C.M., Serebrenik, A., Schiffelers, R.R.H. and van den Brand, Mark G. J. (2018) *Exploring DSL Evolutionary Patterns in Practice - A Study of DSL Evolution in a Large-scale Industrial DSL Repository: .* 2020/12/29. SCITEPRESS - Science and Technology Publications, pp. 446.

Mens, T. and Van Gorp, P. (2006) 'A Taxonomy of Model Transformation', *Electronic Notes in Theoretical Computer Science,* 152, pp. 125-142. doi: 10.1016/j.entcs.2005.10.021.

Mernik, M., Heering, J. and Sloane, A. (2005) 'When and how to develop domain-specific languages', *CSUR,* . doi: 10.1145/1118890.1118892.

Moffat, A. and Zobel, J. (1996) 'Self-indexing inverted files for fast text retrieval', *ACM Transactions on Information Systems,* 14(4), pp. 349-379. doi: 10.1145/237496.237497.

Namavari, A. 'DAWPL: A Simple Rust Based DSL For Algorithmic Composition and Music Production', 1(1), pp. 11.

Ni, W., Ye, K., Sunshine, J., Aldrich, J. and Crane, K. 'SUBSTANCE and STYLE: domain-specific languages for mathematical diagrams', , pp. 2.

Patki, N., Wedge, R. and Veeramachaneni, K. (2016) *The Synthetic Data Vault.* . 10. pp. 399.

Petrovski, B., Aguado, I., Hossmann, A., Baeriswyl, M. and Musat, C. (2018) 'Embedding Individual Table Columns for Resilient SQL Chatbots', *arXiv:1811.00633 cs],* .

Pitchaimalai, S.K., Ordonez, C. and Garcia-Alvarado, C. (2008) *Efficient Distance Computation Using SQL Queries and UDFs.* . 12. pp. 533.

Polosukhin, I. and Skidanov, A. (2018a) 'Neural Program Search: Solving Programming Tasks from Description and Examples', *arXiv:1802.04335 cs],* .

Polosukhin, I. and Skidanov, A. (2018b) 'Neural Program Search: Solving Programming Tasks from Description and Examples', *arXiv:1802.04335 cs],* .

Prabhavalkar, R., Rao, K., Sainath, T., Li, B., Johnson, L. and Jaitly, N. (2017) *A Comparison of Sequence-to-Sequence Models for Speech Recognition.* . 2021/09/12.

Rabiser, R., Thanhofer-Pilisch, J., Vierhauser, M., Grünbacher, P. and Egyed, A. (2018) 'Developing and evolving a DSL-based approach for runtime monitoring of systems of systems', *Automated Software Engineering,* 25(4), pp. 875-915. doi: 10.1007/s10515-018-0241-x.

Ratnaparkhi, A. (1996) *A Maximum Entropy Model for Part-Of-Speech Tagging.* . 2021/04/29.

Reshma, E.U. and Remya, P.C. (2017) *A review of different approaches in natural language interfaces to databases.* . 12. pp. 801.

Riti, P. (2018) 'External DSL', in Riti, P. (ed.) *Practical Scala DSLs: Real-World Applications Using Domain Specific Languages* Berkeley, CA: Apress, pp. 59-69.

Rodrigues, I., Zorzo, A., Da Silveira, M. and de Borba Campos, M. (2018) *Usa-DSL: Usability Evaluation Framework for Domain-Specific Languages.*

Sabour, S., Chan, W. and Norouzi, M. (2019) 'Optimal Completion Distillation for Sequence Learning', *arXiv:1810.01398 cs, stat],* .

Shah, P., Hakkani-Tür, D., Tür, G., Rastogi, A., Bapna, A., Nayak, N. and Heck, L. (2018) 'Building a Conversational Agent Overnight with Dialogue Self-Play', *arXiv:1801.04871 cs],* .

Shah, V., Li, S., Kumar, A. and Saul, L. (2020a) *SpeakQL: Towards Speech-driven Multimodal Querying of Structured Data.* . 06/11; 2021/09/12. Association for Computing Machinery, pp. 2363.

Shah, V., Li, S., Kumar, A. and Saul, L. (2020b) *SpeakQL: Towards Speech-driven Multimodal Querying of Structured Data.* . 06/11; 2022/02/15. ACM, pp. 2363.

Shi, T., Tatwawadi, K., Chakrabarti, K., Mao, Y., Polozov, O. and Chen, W. (2018) 'IncSQL: Training Incremental Text-to-SQL Parsers with Non-Deterministic Oracles', *arXiv:1809.05054 cs],* .

Shin, R. (2019) 'Encoding Database Schemas with Relation-Aware Self-Attention for Text-to-SQL Parsers', *arXiv:1906.11790 cs, stat],* .

Sibuya, M., Fujisaki, T. and Takao, Y. (1978) 'Noun-Phrase Model and Natural Query Language', *IBM Journal of Research and Development,* 22(5), pp. 533-540. doi: 10.1147/rd.225.0533.

Siewiorek, D.P., Hudak, J.J., Suh, B.-. and Segal, Z. (1993) *Development of a benchmark to measure system robustness.* . 06. pp. 88.

Sim, S.E., Easterbrook, S. and Holt, R.C. (2003) *Using benchmarking to advance research: a challenge to software engineering.* . 05. pp. 74.

Simpkins, C., Rugaber, S. and Isbell, C. 'DSL Design for Reinforcement Learning Agents', , pp. 2.

Skeggs, R. and Lauria, S. (2019) 'A Shallow Parsing Approach to Natural Language Queries of a Database', *Journal of Software Engineering and Applications,* 12, pp. 365-382. doi: 10.4236/jsea.2019.129022.

Sobernig, S. (2020) 'A Story of a DSL Family', in Sobernig, S. (ed.) *Variable Domain-specific Software Languages with DjDSL: Design and Implementation* Cham: Springer International Publishing, pp. 261-283.

Soru, T., Marx, E., Valdestilhas, A., Esteves, D., Moussallem, D. and Publio, G. (2018) 'Neural Machine Translation for Query Construction and Composition', *arXiv:1806.10478 cs],* .

Spinellis, D. (2001) 'Notable design patterns for domain-speci®c languages', , pp. 9.

Sriram, A., Jun, H., Satheesh, S. and Coates, A. (2017) 'Cold Fusion: Training Seq2Seq Models Together with Language Models', *arXiv:1708.06426 cs],* .

Starynkevitch, B. (2011) 'MELT - a Translated Domain Specific Language Embedded in the GCC Compiler', *Electronic Proceedings in Theoretical Computer Science,* 66. doi: 10.4204/EPTCS.66.6.

Störl, U., Hauf, T., Klettke, M. and Scherzinger, S. (2015) *Schemaless nosql data stores - object-nosql mappers to the rescue?* Gesellschaft für Informatik e.V.

Su, Y., Hassan Awadallah, A., Wang, M. and White, R.W. (2018) *Natural Language Interfaces with Fine-Grained User Interaction: A Case Study on Web APIs.* . 06/27; 2022/02/15. Association for Computing Machinery, pp. 855.

Suhr, A., Iyer, S. and Artzi, Y. (2018) 'Learning to Map Context-Dependent Sentences to Executable Formal Queries', *arXiv:1804.06868 cs],* .

Sujeeth, A.K., Gibbons, A., Brown, K.J., Lee, H., Rompf, T., Odersky, M. and Olukotun, K. *Forge: Generating a High Performance DSL Implementation from a Declarative Specification.*

Şutîi, A.M., Brand, M.v.d. and Verhoeff, T. (2018) 'Exploration of modularity and reusability of domain-specific languages: an expression DSL in MetaMod', *Computer Languages, Systems & Structures,* 51, pp. 48-70. doi: 10.1016/j.cl.2017.07.004.

Swamy, N., Rastogi, A., Fromherz, A., Merigoux, D., Ahman, D. and Martínez, G. (2020) 'SteelCore: an extensible concurrent separation logic for effectful dependently typed programs', *Proceedings of the ACM on Programming Languages,* 4, pp. 1-30. doi: 10.1145/3409003.

Taghipour, K. and Ng, H.T. (2015) *One Million Sense-Tagged Instances for Word Sense Disambiguation and Induction.* . 07; 2021/04/29. Association for Computational Linguistics, pp. 338.

Talmor, A. and Berant, J. (2018a) 'The Web as a Knowledge-base for Answering Complex Questions', *arXiv:1803.06643 cs],* .

Talmor, A. and Berant, J. (2018b) 'The Web as a Knowledge-base for Answering Complex Questions', *arXiv:1803.06643 cs],* .

Thanhofer-Pilisch, J., Lang, A., Vierhauser, M. and Rabiser, R. (2017) *A Systematic Mapping Study on DSL Evolution.* . 08. pp. 149.

Tisi, M. and Cheng, Z. (2018) *CoqTL: an Internal DSL for Model Transformation in Coq.* . 06/25; 2020/12/27. Springer, pp. 142.

Tratt, L. (2008) 'Evolving a DSL Implementation', in Lämmel, R., Visser, J. and Saraiva, J. (eds.) *Generative and Transformational Techniques in Software Engineering II* Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 425-441.

Utama, P., Weir, N., Basik, F., Binnig, C., Cetintemel, U., Hättasch, B., Ilkhechi, A., Ramaswamy, S. and Usta, A. (2018) 'An End-to-end Neural Natural Language Interface for Databases', *arXiv:1804.00401 cs],* .

Vinyals, O., Bengio, S. and Kudlur, M. (2016) 'Order Matters: Sequence to sequence for sets', *arXiv:1511.06391 cs, stat],* .

Vinyals, O., Fortunato, M. and Jaitly, N. (2017) 'Pointer Networks', *arXiv:1506.03134 cs, stat],* .

Vinyals, O., Fortunato, M. and Jaitly, N. (2015) *Pointer Networks.* . 2022/02/15. Curran Associates, Inc, .

Visser, E. (2007) *WebDSL: A Case Study in Domain-Specific Language Engineering.*

Voorhees, E.M. (2002) *The Philosophy of Information Retrieval Evaluation.* Springer, pp. 355.

Voorhees, E.M. (2001a) 'The TREC question answering track', *Natural Language Engineering,* 7(4), pp. 361-378. doi: 10.1017/S1351324901002789.

Voorhees, E.M. (2001b) 'The TREC question answering track', *Natural Language Engineering,* 7(4), pp. 361-378. doi: 10.1017/S1351324901002789.

Wang, C., Tatwawadi, K., Brockschmidt, M., Huang, P., Mao, Y., Polozov, O. and Singh, R. (2018) 'Robust Text-to-SQL Generation with Execution-Guided Decoding', *arXiv:1807.03100 cs],* .

Wang, C., Brockschmidt, M. and Singh, R. (2018) 'Pointing out SQL queries from text', *ICLR 2018 Conference,* .

Wang, W., Zhang, M., Chen, G., Jagadish, H.V., Ooi, B.C. and Tan, K. (2016) 'Database Meets Deep Learning: Challenges and Opportunities', *ACM SIGMOD Record,* 45(2), pp. 17-22. doi: 10.1145/3003665.3003669.

Wang, W., Tian, Y., Wang, H. and Ku, W. (2020) *A Natural Language Interface for Database: Achieving Transfer-learnability Using Adversarial Method for Question Understanding.* . 04. pp. 97.

Wang, W., Tian, Y., Xiong, H., Wang, H. and Ku, W. (2018) 'A Transfer-Learnable Natural Language Interface for Databases', *arXiv:1809.02649 cs],* .

Weir, N., Utama, P., Galakatos, A., Crotty, A., Ilkhechi, A., Ramaswamy, S., Bhushan, R., Geisler, N., Hättasch, B., Eger, S., Cetintemel, U. and Binnig, C. (2020) *DBPal: A Fully Pluggable NL2SQL Training Pipeline.* . 06/11; 2022/02/15. Association for Computing Machinery, pp. 2347.

Weiss, R.J., Chorowski, J., Jaitly, N., Wu, Y. and Chen, Z. (2017) *Sequence-to-Sequence Models Can Directly Translate Foreign Speech.* Available at: https://ui.adsabs.harvard.edu/abs/2017arXiv170308581W (Accessed: .

Whaley, J. (2000) *A portable sampling-based profiler for Java virtual machines.* . 06/03; 2021/04/29. Association for Computing Machinery, pp. 78.

Wider, A. (2014) *Implementing a Bidirectional Model Transformation Language as an Internal DSL in Scala.* pp. 63.

Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J. and Schwinger, W. (2012) *Fact or Fiction – Reuse in Rule-Based Model-to-Model Transformation Languages.* Springer Berlin Heidelberg, pp. 280.

Winkler, W.E. (1999) *The State of Record Linkage and Current Research Problems.* Statistical Research Division, U.S. Census Bureau. Available at:(Accessed: .

Xu, K., Wu, L., Wang, Z., Feng, Y., Witbrock, M. and Sheinin, V. (2018) 'Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks', *arXiv:1804.00823 cs, stat],* .

Yao, Z., Li, X., Gao, J., Sadler, B. and Sun, H. (2019) 'Interactive Semantic Parsing for If-Then Recipes via Hierarchical Reinforcement Learning', *Proceedings of the AAAI Conference on Artificial Intelligence,* 33(01), pp. 2547-2554. doi: 10.1609/aaai.v33i01.33012547.

Yavuz, S., Gur, I., Su, Y. and Yan, X. (2018) *What It Takes to Achieve 100% Condition Accuracy on WikiSQL.* . 10; 2022/02/15. Association for Computational Linguistics, pp. 1702.

Ye, H., Li, W. and Wang, L. (2019) 'Jointly Learning Semantic Parser and Natural Language Generator via Dual Information Maximization', *arXiv:1906.00575 cs],* .

Yin, P. and Neubig, G. (2019) *Reranking for Neural Semantic Parsing.* . 07; 2021/09/12. Association for Computational Linguistics, pp. 4553.

Yin, P. and Neubig, G. (2018) 'TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation', *arXiv:1810.02720 cs],* .

Yin, P., Zhou, C., He, J. and Neubig, G. (2018) 'StructVAE: Tree-structured Latent Variable Models for Semi-supervised Semantic Parsing', *arXiv:1806.07832 cs],* .

Yu, T., Li, Z., Zhang, Z., Zhang, R. and Radev, D. (2018) 'TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation', *arXiv:1804.09769 cs],* .

Yu, T., Yasunaga, M., Yang, K., Zhang, R., Wang, D., Li, Z. and Radev, D. (2018) 'SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-DomainText-to-SQL Task', *arXiv:1810.05237 cs],* .

Zavershynskyi, M., Skidanov, A. and Polosukhin, I. (2018) 'NAPS: Natural Program Synthesis Dataset', *arXiv:1807.03168 cs, stat],* .

Zdun, U. and Strembeck, M. 'Reusable Architectural Decisions for DSL Design', , pp. 37.

Zhang, X., Yin, F., Ma, G., Ge, B. and Xiao, W. (2020) 'M-SQL: Multi-Task Representation Learning for Single-Table Text2sql Generation', *IEEE Access,* 8, pp. 43156-43167. doi: 10.1109/ACCESS.2020.2977613.

Zhao, C. and Sahni, S. (2019) 'String correction using the Damerau-Levenshtein distance', *BMC Bioinformatics,* 20(11), pp. 277. doi: 10.1186/s12859-019-2819-0.

Zhao, T. and Huang, X. (2018) 'Design and implementation of DeepDSL: A DSL for deep learning', *Computer Languages, Systems & Structures,* 54, pp. 39-70. doi: 10.1016/j.cl.2018.04.004.

Zhong, V., Xiong, C. and Socher, R. (2017a) 'Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning', *arXiv:1709.00103 cs],* .

Zhong, V., Xiong, C. and Socher, R. (2017b) 'Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning', *arXiv:1709.00103 cs],* .