TRANSQLATE: TRANSLATING ENRICHED NATURAL LANGUAGE SENTENCES TO SQL QUERIES USING TRANSFORMERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Mousa Farshkar Azari
September 2022

TranSQLate: Translating Enriched Natural Language Sentences to SQL Queries Using Transformers
By Mousa Farshkar Azari
September 2022

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Özgür Ulusoy(Advisor)

İbrahim Körpeoğlu

İsmail Sengör Altıngövde

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

TRANSQLATE: TRANSLATING ENRICHED NATURAL LANGUAGE SENTENCES TO SQL QUERIES USING TRANSFORMERS

Mousa Farshkar Azari
M.S. in Computer Engineering
Advisor: Özgür Ulusoy
September 2022

A large amount of the structured data owned by different enterprises is typically stored in Relational Database Management Systems, and a decent knowledge of Structured Language Query (SQL) is required to extract desired information from the relational databases. Many naive users need to access the information from databases, and they do not have the necessary skills or knowledge. Additionally, even some expert users might find it challenging to provide complex SQL queries when they do not know the schema underlying the database. To this end, a considerable amount of research has been conducted recently for the translation of queries formulated by users in a natural language to SQL queries to be processed by database systems. In this thesis, we provide some deep intelligent strategies to be used in natural language to SQL translation. We propose TranSQLate, a novel method to enrich the input sequences and provide more effective Natural Language Interface to Database (NLIDB) systems. We apply our strategies to the Vanilla transformer and T5 transformer models in three different ways. With enriched inputs, we achieve up to 16.7% improvement in translation accuracy, 6.5 points in SacreBLEU score, and 18 points in the n-gram precision, compared to not enriched versions. Our method surpasses the strategies used in the state-of-the-art systems NALIR, TEMPLAR, and DBTagger, in terms of translation accuracy over IMDB, scholar, and Yelp datasets.

Keywords: Natural Language Processing, Structured Query Language, Relational Database Systems, Natural Language Interface to Databases, Neural Networks, Deep Learning.

ÖZET

TRANSQLATE: Zenginleştirilmiş Doğal Dil Cümlelerini Transformerler Kullanarak SQL Sorgularina Çevirme

Mousa Farshkar Azari
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Özgür Ulusoy
Eylül 2022

Farklı uygulamalara ait organizasyonların sahip olduğu yapılandırılmış verilerin büyük bir kısmı tipik olarak İlişkisel Veritabanı Yönetim Sistemlerinde depolanır ve ilişkisel veritabanlarından istenen bilgileri çıkarmak için iyi bir Yapılandırılmış Dil Sorgusu (SQL) bilgisi gerekir. Veritabanlarında yer alan bilgilere erişmesi gereken sıradan kullanıcılar erişim için gerekli beceri ve bilgiye sahip değildirler. Ayrıca, bazı uzman kullanıcılar bile veritabanını oluşturan şematik yapıları yeterince bilmedikleri durumda karmaşık SQL sorguları oluşturmayı zor bulabilirler. Bu durum sonucunda, son zamanlarda kullanıcılar tarafından doğal dilde formüle edilen sorguların, veritabanı sistemleri tarafından işlenecek SQL sorgularına çevrilmesi için önemli miktarda araştırma yapılmaktadır. Bu tezde, doğal dilden SQL'e çeviride kullanılacak bazı derin akıllı stratejiler sunuyoruz. Veritabanları için Doğal Dil Arayüzü (NLIDB) sistemlerini daha etkin duruma getirebilmek amacıyla yeni bir yöntem olan TranSQLate'i öneriyoruz. Stratejilerimizi Vanilla ve T5 dönüştürücü modellerine üç farklı şekilde uyguluyoruz. Zenginleştirilmiş girdilerle, zenginleştirilmemiş versiyonlara kıyasla çeviri doğruluğunda %16,7, SacreBLEU skorunda 6,5 puan ve n-gram hassasiyetinde 18 puana varan iyileşme elde ediyoruz. Önerdiğimiz yöntem, IMDB, scholar ve Yelp veri kümeleri üzerinde çeviri doğruluğu açısından, son teknoloji sistemler NALIR, TEMPLAR ve DBTagger'da kullanılan stratejilere üstünlük sağlamaktadır.

Anahtar sözcükler: Doğal Dil İşleme, Yapılandırılmış Sorgu Dili, İlişkisel Veritabanı Sistemleri, Veritabanları için Doğal Dil Arayüzü, Sinir Ağları, Derin Öğrenme.

Acknowledgement

I would like to express my most sincere gratitude and appreciation to my advisor Prof. Dr. Özgür Ulusoy for his endless support, motivation, and patience during my master's studies. The completion of this thesis would never be possible without his help.

I would like to thank Dr. Arif Usta for his exceptional guidance and help in completing my thesis with his remarkable knowledge in the field.

I am also thankful to the jury members for their time and valuable suggestions on my thesis: Prof. Dr. İbrahim Körpeoğlu and Assoc. Prof. Dr. İsmail Sengör Altıngövde. Special thanks to the Bilkent University Computer Engineering Department along with the Scientific and Technical Research Council of Turkey (TÜBITAK) for their financial support under grant number 118E724.

I am grateful for all my friends who always made life more enjoyable with their friendship and support through good times and hard times. During my studies, I had the chance to learn a lot from my dear friend Salman. I would like to appreciate all his help.

Last but not least, I express my deepest gratitude to my beloved family for their continued love and support. I cannot thank enough my brother, Alireza, for helping our family during my studies. I am forever grateful for my caring, patient, and supportive family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The significance of database systems has increased drastically in recent years due to a large amount of data being stored in them. Relational databases are the most popular databases which store the data in a structured form. The standard way to access the information stored in relational databases is to use structured query language (SQL). Writing a SQL query might be challenging for most naive users who need to extract data from the relational database and do not have the required knowledge. Additionally, the expert users may also find it difficult to write some complex queries or they might spend too much time finding the correct way to express the SQL query. The user also needs to understand the whole structure of the database schema to be able to provide the right SQL query. Due to these limitations, using SQL queries is less preferred.

To resolve the constraints, the researchers of the database community came up with a promising solution which is natural language interfaces to databases (NLIDBs) [1]. NLIDBs provide a user-friendly interface similar to search engines which allow users to retrieve data from relational databases without any SQL knowledge. With the emergence of the NLIDB systems, the user just needs to write a natural language query in the desired language and the NLIDB system will translate the provided natural language into a SQL query. The SQL query is then used to extract the requested information from the database and show the

final results to the user.

There are two different approaches when translating natural language to SQL query; end-to-end strategies [40, 44] and pipeline-based solutions [41, 31]. Deep learning methods have been widely utilized in both of these approaches.

1.1 Motivation and Contributions

Most of the recent studies have employed deep learning techniques to provide faster, more effective, and user-friendly NLIDBs [32, 9, 39]. Although the NLIDB systems have become the research hot-spot in the database community, there is still room for improvement regarding some state-of-the-art work in this research area. In this thesis, we propose a novel solution to improve the translation quality in NLIDB systems.

Various approaches have been proposed to improve the translation accuracy but most of these works are utilizing complex deep learning models which usually require a long training time along with a substantial amount of data. One of the most effective approaches to boost the current results is to enrich the input data and feed the enriched input sequence to the model [42, 18, 11]. We propose TranSQLate, a novel method to enrich the input sequence and achieve better results. We apply TranSQLate over two different transformer models (Vanilla and pre-trained T5) that we have developed, and we achieve up to 16.7% improvement in accuracy and 6.5 points in SacreBLEU score [26]. Our experiments have been performed on IMDB, Yelp [24], and scholar [16] datasets. TranSQLate is a simple yet effective method that can be applied to any model to surpass its own results.

The work proposed in this thesis is based on the outputs of DBTagger [36, 37] which is used in NLIDB systems for the keyword mapping task. DBTagger formulates the keyword mapping task as a sequence tagging problem and offers a unique supervised deep learning solution that uses POS tags from natural language queries. DBTagger is a schema-independent method and it is applicable in

different relational databases. DBTagger outputs include schema tags and type tags which have been utilized in the enrichment step of our proposed model.

The main contributions of our proposed model are as follows:

- **Input Enrichment**: We propose a novel method (TranSQLate) to enrich the input sequence to help the model to make a better and more accurate prediction which helps to improve the overall translation accuracy. The enrichment task is done by taking two additional features into account.
- **Transformer-Based NLIDB models**: We have developed two different NLIDB models (Vanilla and pre-trained T5) for translation from natural language to SQL query. Both of our models are based on transformer architecture. The generated SQL query can be submitted to the database to extract the desired information. These models are generic and extendable to other text-to-text NLP problems.
- **Generic Model and Method**: Since the enrichment part is performed over the input data, we can apply each of the strategies proposed by TranSQLate on any other model which translates text to SQL. Furthermore, the pre-trained T5 model is also generic and can be extended to various text-to-text problems in the natural language processing domains such as question answering, text summarization, neural machine translation, etc.
- **Performance Boost**: With our proposed methods, we achieve up to a 16.7% improvement in translation accuracy, 6.5 points in SacreBLEU score, and 18 points in the n-gram precisions, compared to not enriched versions.

1.2 Outline

Following the introduction chapter, we go through the related work done in the literature in Chapter 2. In Chapter 3, we provide a comprehensive background regarding the methods and the architectures that we use in this work. We explore the methodology of our proposed work in Chapter 4 where we provide a

detailed and step-by-step illustration of each different strategy proposed to enrich the natural language queries. Next, in Chapter 5, we present the experimental evaluation results of our work. Finally, in Chapter 6 we provide the conclusions and prospective future work.

Chapter 2

Related Work

In this section, we present the related work first on Natural Language Interface for Database (NLIDB) systems, and then Transformers and the approaches used to enrich the input data in NLIDB systems. In each section, we briefly go through what our work contributes to each of the topics.

2.1 Natural Language Interface to Databases (NLIDBs)

An NLIDB system allows users without technical knowledge to effortlessly retrieve information from relational databases by translating a natural language query (NLQ) provided by the naive user to the relevant SQL query and extracting the required information from the database. NLIDB systems have been the research hot-spot in the database community over the past years. Pattern-matching, constructed grammars and guidelines, syntactic parsers, and intermediate language representations were all used in early studies, that were frequently used in a multi-stage setup [1]. Modern techniques, on the other hand, address the NLIDB challenge as a sequence-to-sequence problem. A massive end-to-end neural network receives the input sequence and produces the output in this scenario. This

method eliminates the need to learn and combine different language components.

Seq2SQL [44] encodes a sequence of columns from the corresponding table, SQL keywords, and a query using bidirectional LSTM. It produces three distinct components: an aggregation classifier, a select column pointer, and a where clause. One disadvantage of this technique is that the select column pointer has just a single output. To answer a question, various columns are usually necessary. Moreover, the authors' approach mainly focuses on value mapping while ignoring table and attribute mapping.

SQLNet [40] presents a sequence-to-set approach and also a technique for column attention. The model calculates a score for each column in order to choose the columns that will appear where clause thresholding is applied. This model has the same weakness as Seq2SQL since the output structure is the same. In addition, both of these models have been tested on the WikiSQL dataset which has a single table. For most relational databases, such a configuration is not a realistic choice.

NALIR [17] is an NLIDB system that relies on parsing. The system first parses the input with the Standford Parser [7] and then extracts the dependency tree exclusively from it. Nodes represent any words or phrases that can be mapped to SQL components in the linguistic parse tree. It aims to map keywords into potential database schema items using two different similarity measures, meaning and spelling.

Baik et al. [3] propose TEMPLAR which utilizes SQL query log information to improve the translation accuracy. TEMPLAR is applied over an existing NLIDB where it utilizes query logs to find database fragments that will be mapped to query keywords for database value matching later on. It also utilizes a similarity model, such as word2vec, to estimate similarities between keywords and database components before mapping them.

2.2 Input Enrichment

TypeSQL [42] architecture includes two Bidirectional Long Short-Term Memories (BI-LSTMs). The authors use the type information as an extra input layer. They concatenate the embeddings of words and their relevant types, and then they feed it to the BI-LSTM encoder. Type information is obtained after tokenizing the input sequence when they search over the table schema. They provide different categories for tagging the numbers that appear in the input sequence but for named entities they utilize freebase to assign them to different categories. This work is limited to the type information in ten categories and it is not taking full advantage of schema information. Additionally, unlike most of the state-of-the-art work which is based on transformer architectures, they use BI-LSTM which has shorter memory for the words in the input sequence that each word can refer to.

BRIDGE [19] provides the natural language query and the Database schema as a tagged sequence in which a portion of the fields is enriched with cell values from the natural language query. BRIDGE model is based on BERT. In order to enrich the input, after reaching the end-of-sequence token, the type information such as table name and field name is being appended to the same sequence in between some special tokens. After adding this information to each input sequence, another end-of-sequence token is added to the end of the same sequence. The results show that this method of enrichment boosts the results. In the BRIDGE model, BERT encodes the enriched sequence with a few additional layers, and contextualization of the input text and the database is accomplished with fine-tuned deep attention in BERT. The model is paired with a pointer-generator decoder that uses schema consistency to limit the searching space.

IRNet model [11] decomposes natural language into three parts, rather than synthesizing SQL queries in an end-to-end manner. The proposed method provides schema linking across a query and database schema. Then, the model uses a grammar-based neural network to generate an intermediate representation designed to bridge the gap between natural language query and SQL. Eventually,

IRNet uses domain knowledge to predictably infer a SQL query from the synthesized same intermediate representation. IRNet leverages SemQL, a domain-specific language that acts as a bridge between SQL and natural language. The model architecture includes an encoder for natural language queries, a schema encoder, and a decoder. Furthermore, IRNet defines three different type tags for tables, columns, and values in the database. After detecting the type tags, they are fed to a schema encoder in the IRNet model (the schema encoder accepts a database schema as input and returns column and table representations) and the type information is used to help the model to make better predictions for the generated SQL query.

GAP [34] takes the natural language query enriched with column names in the schema in a different format. Unlike BRIDGE, there is only one end-of-sequence token and the column names are fed to the model inside some special tokens defined for this task. GAP uses generation models to generate pre-train data to simultaneously learn representations of natural language sequences and table schema. GAP model is trained on 2 million NLQ-schema pairs and 30 thousand NLQ-schema-SQL triples created using generative models. To enrich the input, it concatenates the input sequence with relevant column names in the used schema.

EditSQL [43] is primarily concerned with the cross-domain context-dependent text-to-SQL problem. The authors make use of the fact that neighboring natural language queries are dependent on each other and that matching SQL queries overlap. They use this dependence to improve the generation quality by modifying the previously anticipated query. The editing mechanism accepts SQL sequences as input and reuses generating outputs at the token level.

To manage more complicated tables in many domains, an utterance-table encoder and a table-aware decoder are utilized to include the natural language context and the schema. User utterances and table schema are encoded using the utterance-table encoder. To encode sentence tokens, a bi-LSTM is utilized. To choose the most relevant columns for every word, an attention-weighted average of column header embedding is employed. To record the connection between the table schema and the utterance, an attention layer is included. An interaction-level

decoder is developed on top of the utterance-level encoder to capture information from various utterances. The conversion concludes with the construction of SQL queries utilizing an LSTM decoder to incorporate interaction history, table schema, and user utterance. The enrichment method is very similar to the one used in BRIDGE.

The RAT-SQL architecture [39] is built on the relation-aware self-attention technique, which enables address schema encoding, feature representation, and schema linking inside a text2SQL encoder. Different from many models that utilize schema encoding, RAT-SQL focuses more on schema linking. Schema linking associations assist the model match column/table references in the query with the relevant schema columns/tables. Two types of information in the input sequence are used for schema linking: Name-Based Linking where they check if a table or a column name has appeared in the input sequence, and Value-Based Linking where the natural language query contains any values found in the database and so participating in the expected SQL.

Chapter 3

Background

NLIDB systems, which are being designed to take user queries in natural language (NL), and then translate this NL to a SQL query, are receiving significant interest lately. The SQL query is then processed to retrieve the associated information from the database. This Text-to-SQL challenge is a long-standing, open research problem, and the typical method of tackling it is to construct a sequence-to-sequence model. In this chapter, we go through the background work done prior to our work and we provide a detailed explanation of each related topic. The background chapter is based on the different transformer architectures that we use, and the pre-trained language models that we obtain our additional input features from.

3.1 Deep Learning Architectures in NLP

The sequence-to-sequence model structure usually contains an encoder and a decoder and they are the standard way to represent text sequences. The encoder-decoder architecture is developed to take an entire input sequence and get an internal representation of it. This internal representation has a fixed length and later it is fed to the decoder. The decoder uses this information to generate words

until an end-of-sequence token is generated. For instance, suppose that the following natural language query: "Find all movies produced in 2015" is fed to a sequence-to-sequence model with the encoder-decoder architecture. The encoder will process the input token by token and transform it into the internal representation state, and then pass it to the decoder to generate the target sequence (which is again token by token): "select movie_0.title from movie as movie_0 where movie_0.release_year = 2015".

The RNN-based [6] encoder-decoder models process the input sequence piece by piece, and while working with a huge amount of data this may turn into a real problem. Additionally, for a word W in the input sequence, RNNs have a short and limited memory of the other words that W can refer to. This is another drawback while using RNN-based models. In 2017, the very first transformer [38] was introduced to address these problems. Various Deep Learning applications, including Natural Language Processing (NLP), Neural Machine Translation (NMT), Question Answering (QA), Text Summarization, and recently Computer Vision (CV) and Audio Processing, have demonstrated significant performance improvement using transformers. Almost all of the cutting-edge NLP performances are based on transformer structures. Transformers were initially introduced in the *Attention Is All You Need* [38], and since then, plenty of other versions called as X-formers have developed. All of the X-formers are based on the vanilla transformer architecture by trying to improve it for certain usage. The next section provides an in-depth explanation of the transformers.

We can categorize the transformers into three different groups as follows.

- **Encoder Only**: Models with this structure implement only the encoder architecture of the transformer and they are usually used for natural language understanding, classification, or sequence labeling problems. BERT [8] and RoBERTa [20] are the most popular examples of this category. The difference between BERT and RoBERTa is the next sentence prediction feature which does not exist in RoBERTa since the authors found the performance not good for the downstream tasks.

- **Decoder Only**: Models with this structure have just utilized the decoder part of the transformer architecture so they do not include the cross-attention between the encoder and the decoder. These models have been widely used for text generation tasks. GPT [27] and GPT-2 [28] are the most popular decoder-based transformers which show state-of-the-art results for text generation tasks.

- **Encoder-Decoder**: In this case, the full transformer architecture is utilized. Such transformers are usually used in sequence to sequence models. BART [15] which extends the denoising mechanism in BERT [8] to the encoder-decoder architecture, and T5 [29] which used a similar design and was among the first experiments to employ task-specific text prefixes in downstream tasks fall in this category. The most popular usage of this architecture is neural machine translation (NMT) and we adapt this architecture for developing our models.

3.2 Transformers

One of the most important components of the transformers is the attention mechanism [22, 2] which is designed to allow the decoder to use the most important sections of the input sequence in a functional way by using a weighted combination of all of the encoded input vectors, while the most relevant vectors obtaining the greatest weights. Our vanilla transformer model has been implemented from the scratch and it is based on an encoder-decoder structure [6]. The encoder maps the input sequence into an abstract continuous representation that contains all of the input's learned information. The decoder receives the continuous representation and creates a single output step by step while also being fed by the previous output recurrently until an "end of sequence" token (which is $< end >$ in our case) is generated. Figure 3.1 illustrates the vanilla transformer architecture including all components. In the following subsections, we study each part of the transformer architecture in detail which allows us to have a step-by-step understanding of how the output is generated based on a given input.

3.2.1 Embeddings and Positional Encoding

The initial step is to send our data to a word embedding layer. In this model, we got the learned representations of the tokens from *fasttext* [5, 12] where each token is mapped to a vector with continuous values to represent that token.

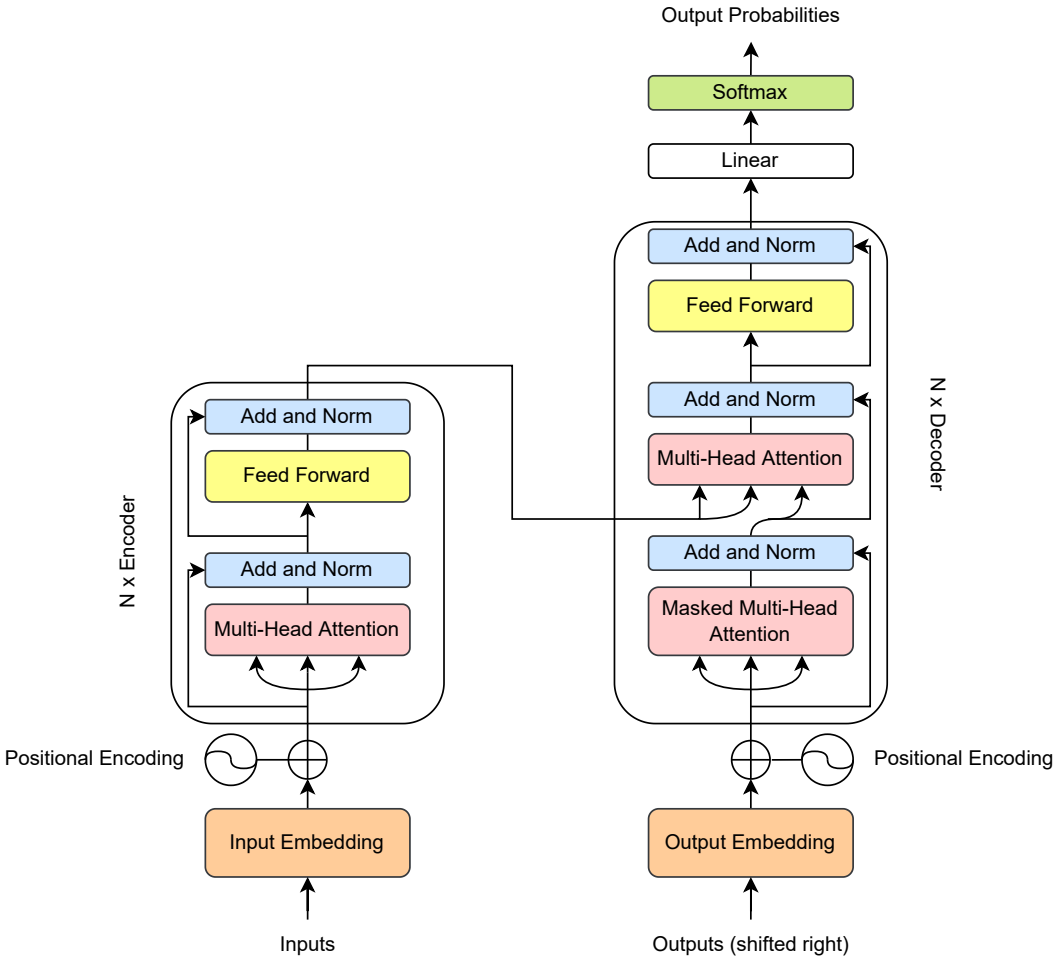

Figure 3.1: The architecture of the Vanilla transformer model

The positional information is then injected into the embeddings in the following step. Unlike the recurrent neural networks [6] the transformer encoder has no recurrence, therefore, we need to provide details for each token's absolute position for the embeddings. The positional encodings and embeddings have the same dimension d, therefore the two can be summed. This step is done using positional encoding which for every odd time step, creates a vector using the cosine function,

and for every even time step, creates a vector using the sine function:

$$PositionalEncoding(position, 2d+1) = \cos\left(\frac{position}{10000^{2d/e_{model}}}\right) \tag{3.1}$$

$$PositionalEncoding(position, 2d) = \sin\left(\frac{position}{10000^{2d/e_{model}}}\right) \tag{3.2}$$

where *position* is the position, d is the dimension, and e_{model} is the size of the embedding layer.

These vectors are added to their corresponding embedding vector. This effectively feeds the network about the positions of each vector.

3.2.2 Encoder and Decoder

The transformer encoder is a stack of encoders in layers. Stacking the encoders allows the transformer to process the sequence not only once but a couple of times. The decoder also has the same architecture. The idea behind stacking is to be able to capture the raw properties of the sentence first moving to more elaborate properties. It has been shown that these layers inside the stack are discovering the NLP pipeline [35]. The first layers focus on identifying part-of-speech tags, followed by constituents, dependencies, semantic roles, and coreference.

Inside the encoder stack, there is a residual connection, which means the output of each layer is added to the output of the next layer. This mechanism ensures that no information is lost by any layer without any chance for the next layer to recover those lost data.

The encoder layer's task is to map all input sequences into an abstract continuous representation that contains all of the learned information for the whole sequence. It contains two sub-modules, multi-headed attention followed by a fully connected network. There are also residual connections around each of the sub-modules followed by a layer normalization. All encoder layers have the same kind of

structure. The self-attention layer processes the input and transmits it to a feed-forward neural network.

The decoder part of the architecture has the same structure as the encoder with a slight difference which is an encoder-decoder attention layer. The decoder processes the vectors like the encoder but also computes attention between the decoder vectors and the output of the first encoder. This helps the decoder to focus on the more appropriate input parts.

3.2.3 Multi-Headed Attention

The encoder's multi-head attention leverages a special attention mechanism known as self-attention. The model use self-attention to link every individual token in the input sequence with all the other words in the same sequence. We pass the input into three different fully connected layers to form the query, key, and value vectors in order to accomplish self-attention. To generate a score matrix, the queries and keys are multiplied by a dot product operation. The score is calculated as:

$$Score(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{3.3}$$

where Q, K, V are query, key, and value respectively, and d_k is the dimension of the queries and the keys. The score matrix indicates how much emphasis a word should place on the other words, such that each word has a score that corresponds to other words in the time step. This score is directly related to the focus. By utilizing the attention mechanism [38] the model has infinite memory for the previous words in the input sequence to reference from. After the self-attention layer is computed on the input of its output vectors (taking into account the dependencies of words between each other), it passes its vectors onto a simple, feed-forward neural layer. This happens in all encoder layers.

3.2.4 Linear Classifier and Softmax

The last point-wise feed-forward layer's output is passed through a final linear layer that functions as a classifier. The classifier is as large as the number of classes. The classifier's output would then be passed into a softmax layer, which generates probability scores ranging from 0 to 1. The index with the greatest probability score reflects our predicted word.

3.3 Pre-Trained Language Models

Due to the popularity of the transformers along with state-of-the-art results in various downstream tasks of NLP, different variants of the transformers have been developed with a huge amount of data. All of these language models adapt the vanilla transformer architecture and build the model based on that. In this section, we will explore some of these models and briefly explain them.

3.3.1 BERT

BERT [8] model is an encoder-based (or auto-encoding model), that depends on the encoder side of the original transformer, allowing the model to observe all tokens in the attention heads. In pretraining, targets are the actual texts, while inputs are corrupted forms of those texts. BERT modifies the input with a special technique called random masking. The masking is done over a custom percentage (15%) of the inputs using a special mask token, an arbitrary token other than the one masked, or the same token.

3.3.2 GPT-2

GPT-2 [28] is a decoder-based (or auto-regressive) model which is pretrained on WebText, a larger and improved variant of the original GPT model. Decoder-base models depend on the original transformer's decoder and utilize an attention mask, allowing the model to just focus on the tokens before the attention heads at each position. GPT-2 is trained to predict the next token in a sequence. Using this functionality, GPT-2 potentially generates syntactically coherent text. When the model is initialized with random input, GPT-2 generates synthetic text samples. Therefore, it is mostly used for text generation or next word prediction tasks.

3.3.3 BART

BART [15] contains both the encoder and the decoder of the original transformer architecture and it is a sequence-to-sequence model. Similar to BERT, a corrupted input sequence is injected into the encoder while the decoder gets original tokens where the future words are masked. The masking technique is similar to BERT and is applied over a range of "k" tokens. BART is mostly used for conditional generation and sequence classification tasks.

3.3.4 T5

The Text-To-Text Transfer Transformer (T5) [29] is one of the most recent and also most powerful transformer models that has been pre-trained on a huge amount of data. The T5 model has gained state-of-the-art results in a variety of NLP tasks such as machine translation, document summarization, question answering, and classification. The T5 model is not limited to the tasks mentioned in the paper, we can fine-tune it to use in different downstream tasks. T5 is also a sequence-to-sequence model which contains both the encoder and the decoder of the original transformer with a very slight modification in positional embeddings

that are learned at each layer.

The T5 model formulates any NLP task into text-to-text. This means that the input and the output are always text strings while in other transformer-base models like BERT [8] the output can either be a class or a span of the input. This is the most powerful thing about the model. We can fine-tune the model for desired NLP task by just formulating the problem into the text-to-text format without a need to change any of the hyperparameters, loss function, learning rate, or optimizer. The overall architecture of the model is very similar to the structure of the vanilla model shown in Figure 3.1.

They have released five unique checkpoints with different parameter sizes for the pre-trained model. Table 3.1 shows the information about each checkpoint. Our model is based on T5-base.

Table 3.1: Different Checkpoints of the T5 model

model	**parameters**	**# layers**	**# heads**
T5-Small	60 million parameters	6	8
T5-Base	220 million parameters	12	12
T5-Large	770 million parameters	24	16
T5-3B	3 billion parameters	24	32
T5-11B	11 billion parameters	24	128

3.4 DBTagger

DBTagger [37] is a novel supervised learning method based on deep neural networks and defines the keyword mapping problem as a sequence tagging task. DBTagger is an end-to-end solution and at the same time schema independent therefore it can easily be extended to different relational databases. The model uses the POS (part-of-speech) tags of the natural language queries in the sequence tagging problem.

To enrich the input, we take two extra features into account. These features are Type Tags and Schema Tags obtained from DBTagger. A lookup table has been generated for each of the instances in the natural language query and relevant tags. We search for respective tags for each token (which is an instance in a natural language sequence) in the related lookup table and feed the corresponding tag to our network.

Our model contains two extra learnable embedding layers for each of the tags. The embedding size for each of the layers is different. For Type Tags, it is 50 and for Schema Tags, it is 100. We explain these tags in detail in the next (Methodology) chapter.

Chapter 4

Methodology

We have developed two different transformer-based models with our proposed methods. In both of the models, we have used transformer architectures since they are known as state-of-the-art for almost all of the NLP problems. The first model is based on "Attention Is All You Need" [38] architecture. In our second approach, we have used a pre-trained T5 [29] language model. We will call the models Vanilla and pre-trained T5, respectively. In the pre-trained model we used the T5-base tokenizer, and in the Vanilla model we implemented a very simple yet effective tokenizer from scratch. The T5-base tokenizer is based on SentencePiece which supports two segmentation algorithms, byte-pair-encoding (BPE) [33], and the unigram language model [14]. The key with the transformers is that while RNNs [6] process the input piece by piece, the transformer processes the whole input sequence at once. With this parallelization in processing the input sequence, the model gains a lot of power. In the following sections, we describe what methods we use to enrich our models, but before that, we need to explain what tags are used to enrich the input and how we obtain them.

4.1 Mapping Scheme

For the enrichment task, we use two different tags associated with each word in the input sequence. The objective of enriching the input is to feed extra information about the schema and database to the model and allow the decoder to generate more relevant words as the output. We call these additional tags "Type Tag" and "Schema Tag". Figure 4.1 provides an example ER diagram for a subset of the IMDB dataset which would help to understand the idea behind the tags better.

4.1.1 Type Tags

Any natural language query includes keywords that can be mapped to database schema components such as a table, attribute, value, or condition. Type tags indicate the type of mapped schema component that will be utilized in the SQL query. We have seven different type tags overall.

- **TABLE**: This tag is used for the nouns in the natural language query which can have direct references to the tables in the database schema. As illustrated in Table 4.1, the word *movies* is a noun and refers to the table *movie* and as a result, it is labeled with a TABLE tag.
- **TABLEREF**: Similar to the nouns, some verbs could have references to the tables. Many of these tables are relation tables. The example in Table 4.1 includes the word *produced* which is a verb and therefore tagged as TABLEREF.
- **ATTR**: Attributes are extensively used in SQL queries in the SELECT, WHERE, and GROUP BY sections. The natural language query may contain some nouns that can be tagged as attributes. In natural language queries, we apply the ATTR tag to label relevant terms. Revisiting the example provided in Table 4.1, the word *year* is a noun and refers to a column in the table *movie*, therefore, it is labeled with an ATTR tag.

- **ATTRREF**: Similar to the difference between TABLE and TABLEREF tags, the tokens in the natural language query which are verbs and have a reference to an attribute in the SQL query are tagged with the ATTRREF label.

- **VALUE**: Some words or tokens in the natural language query could have a direct reference to the values in the database. The words that define people, places, year, etc. usually fall into this category. For such words, the VALUE tag has been used. Tokens with VALUE tags could potentially be utilized in the translation to identify "where" clauses in SQL. In Table 4.1, we can see that the token *2012* is mapped as a VALUE.

- **COND**: Following the determination of which words in the query are to be mapped as values, it is also necessary to determine the phrases that represent which types of SQL query conditions must be fulfilled. Such tokens are labeled with a COND tag. An example is given in Table 4.1.

- **O (OTHER)**: Usually, there are other words in the natural language query that does not belong to any of the above tags. There is no need to map such words to an element in the database schema. Most of the stop words are tagged with O.

Table 4.1: An example of a natural language query with a corresponding type tag and schema tag for each token at two distinct levels

NL query	**Type tags**	**Schema tags**
How	O	O
many	O	O
movies	TABLE	movie
were	O	O
produced	TABLEREF	O
in	COND	COND
the	O	O
year	ATTR	movie.release_year
2012	VALUE	movie.release_year

4.1.2 Schema Tags

Keyword schema tags describe the database mapping to which the token refers. As mentioned in the previous subsection, they could refer to the name of a table or attribute. Labeling a token with a type tag is necessary but insufficient. An additional labeling layer has been utilized where the tags refer to table names or attributes. The schema tags are defined for every entity table and every attribute which is not a primary key (PK) or foreign key (FK).

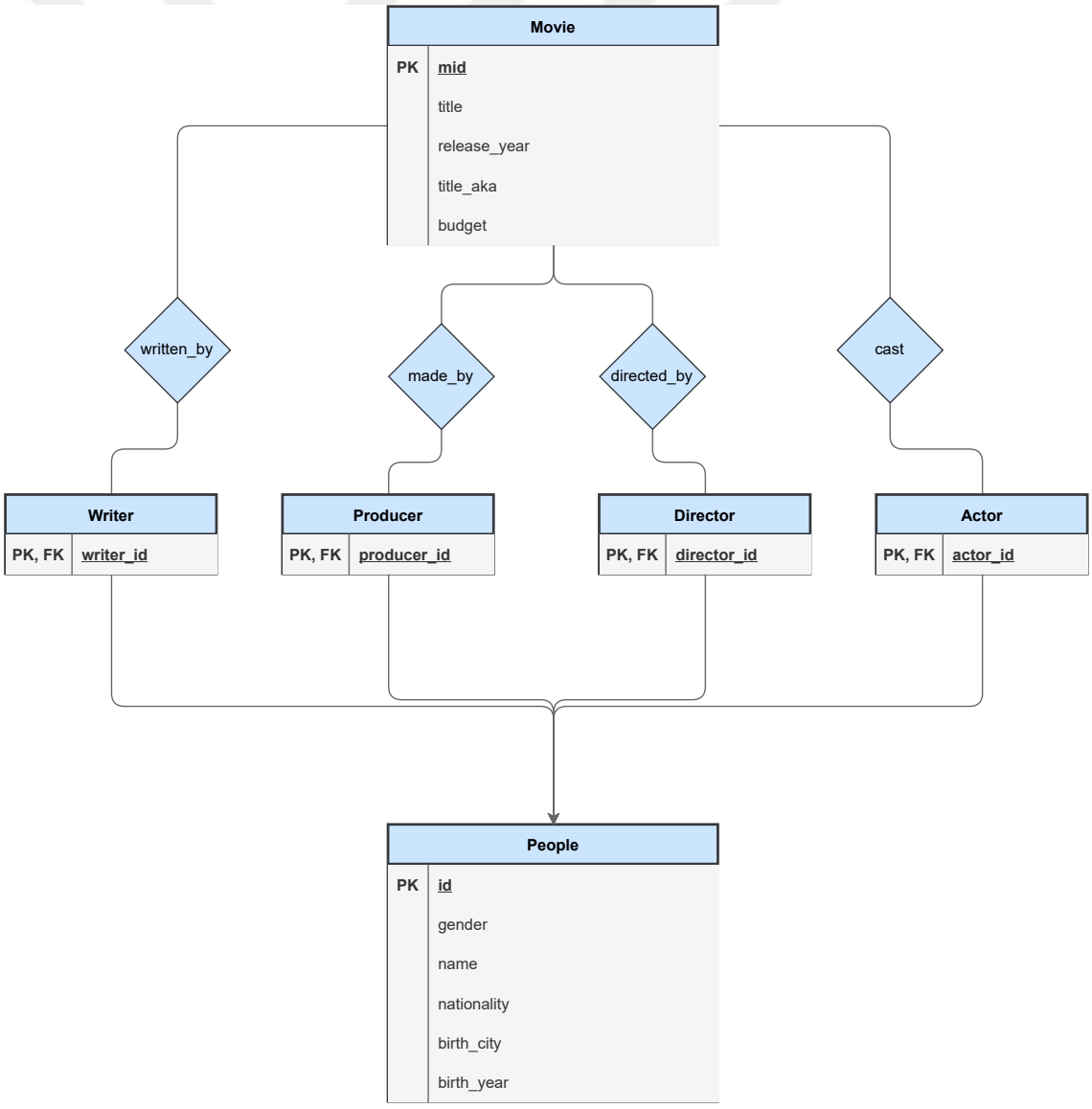

Figure 4.1: An example ER diagram for the IMDB dataset.

4.2 Input Enrichment

Even though both of our models show steady results while feeding the natural language query by itself to the model, we came up with novel methods to even make the results better. In this section, we explain how the enrichment is done in our Vanilla and pre-trained models.

4.2.1 Enrichment in Vanilla Model

In our Vanilla transformer model, we use both type tags and schema tags. We take the natural language query, then we tokenize it and get the embeddings of each token. The embedding layer is inside the encoder of our Vanilla transformer model. For the type tag and schema tags, we initialize two additional input layers inside the encoder. The size of the layers is different from each other which is 50 for the type tags and 100 for the schema tags. We check for the corresponding tag for each token and feed it to the relevant layer. Finally, in the forward function of the encoder, we concatenate the type tags and schema tags with the matching token from the input sequence. Figure 4.2 illustrates the overall flow of the inputs and tags in the Vanilla model. Later, the output of the encoder is passed to the decoder, and the output is generated.

4.2.2 Enrichment in T5 Pre-Trained Model

For enriching the input in the pre-trained model, we take two different approaches and feed the tags along with the input sequence to the model. We call these two approaches v_1 and v_2, respectively. An example of the input enrichment for both v_1 and v_2 provided in Figure 4.3.

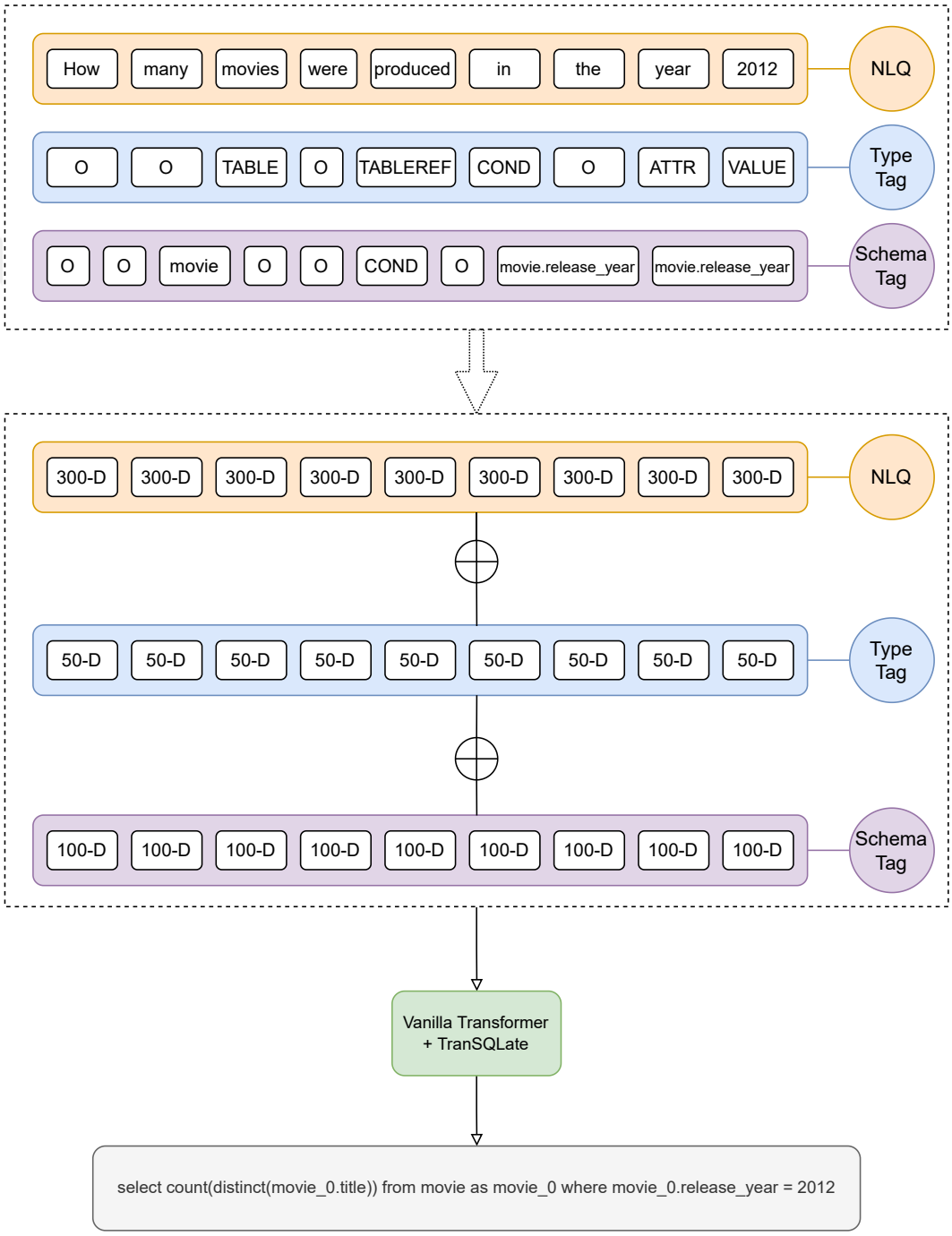

Figure 4.2: An overview of TranSQLate in the Vanilla model

T5+TranSQLate v_1

In our first method (v_1), given a natural language query (NLQ), type tag t, and schema tag s, the input for our model is the concatenation of NLQ, t, and s in the following format:

$$input = \{< start > NLQ < t_1 > s_1 < /t_1 > ... < t_i > s_i < /t_i >< end >\} \tag{4.1}$$

The tags are appended to the input sequence until the i^{th} token. Note that there might be similar tags for different tokens in the input sequence in such cases we only include one of them. According to the input format, we successfully feed all useful elements to the model to improve the results of translation accuracy, SacreBLEU, and n-gram precisions.

T5+TranSQLate v_2

In our second method (v_2), we take the input NLQ which contains i tokens where i^{th} token is shown as w_i. Then we check if any tags exist for a given token w_i from the input sequence and if there is a corresponding tag for that token we append it right next to w_i. Note that the "O" tag in both type tags and schema tags are excluded from the enriching task in any of these methods since it does not provide any useful information for the model. The second method can be illustrated in the following format:

$$input = \{< start > w_1 < t_1 > s_1 < /t_1 > ... \; w_i < t_i > s_i < /t_i >< end >\} \tag{4.2}$$

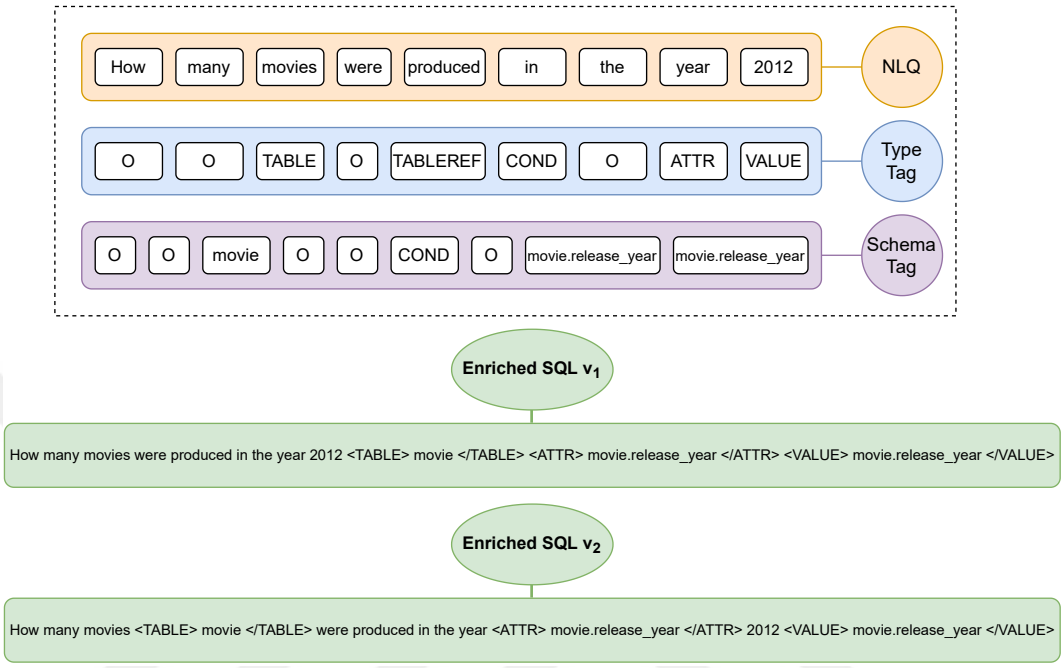

Figure 4.3: An overview of TranSQLate in both versions of the pre-trained T5 model

4.3 Text to SQL Translation with Vanilla Transformer

As the initial step in our Vanilla model, we tokenize the input sequence and obtain the vector representation for each token from fastText [5, 12]. Then we add two extra input layers for type tags and schema tags to the encoder. These new layers are from different sizes and they are concatenated with the vector representation of each token in the forward function of the encoder which is fed into a dropout layer. The self-attention mechanism in the encoder links each token in the natural language query with other tokens in the same sequence which is passed to the feed-forward layer. After every step is done in the encoder, we get an intermediate representation that includes all of the learned information from the encoder side.

The decoder on the other side gets the abstract representation, and like the encoder, it analyzes vectors but also computes attention between the decoder vectors and the output of the first encoder. This allows the decoder to focus

on the most relevant input segments. The steps are similar to the ones in the encoder as shown in Figure 3.1. Both encoder and decoder contain 6 layers and since the embedding size should be divisible by the number of attention heads, we used 15 heads in the vanilla model. To generate the results, the output of the decoder passes through a linear layer followed by a softmax layer.

4.4 Text to SQL Translation with Pre-Trained T5 model

Since the architecture of the pre-trained model is also based on the Vanilla transformer structure with only slight differences, the flow of the data, in general, is similar except for a few parts. In the T5 model, we did not implement our own tokenizer and we use the T5 default tokenizer since we found it the optimum by analyzing the results. The T5 model requires a prefix in each input sequence to determine the task. In our case, we add *Translate English to SQL:* to the beginning of each input. Later, we enrich the natural language queries with each of the methods mentioned in the enrichment section and the rest of the operations are like the Vanilla model. Our model is based on the T5-base checkpoint which contains 220 million parameters, 12 layers, and 12 heads.

4.5 Loss Function

While working with neural networks and deep learning models, in order to tell how good are the results produced by the model a loss function is used. There are many loss functions used for various machine learning problems and the most reliable one for neural machine translation is cross entropy [10]. With cross-entropy, the error is calculated as a value between 0 and 1. A model with 0 loss is considered the ideal one so, the aim is to minimize the loss and get a value as close to zero as possible.

We measure the gap between the actual and the predicted probabilities of both of our models with the cross entropy loss function. Cross entropy can be used on both binary and multi-class classification. This loss function is defined as:

$$Loss_{CE} = -\sum_{i=1}^{n} t_i \log(p_i), \text{ for n classes} \tag{4.3}$$

where t_i is the actual label, p_i is the softmax probability for the i^{th} class, and the log is computed to base 2.

4.6 Optimizer

Optimizers are algorithms or approaches that are used to adjust the properties of the neural network, such as weights and learning rate, in order to minimize loss. For training a deep learning model an optimizer should be used. We have used different optimizers in our models, the Vanilla model utilizes stochastic gradient descent (SGD) [30] whereas the pre-trained model employs an AdamW [21] optimizer. We explain each of these optimizers in the following subsections.

4.6.1 Stochastic Gradient Descent (SGD)

SGD [30] is a straightforward yet highly effective method for fitting linear classifiers and regressors to convex loss functions such as (linear) Support Vector Machines and Logistic Regression. The SGD equation is applied to update attributes in a deep learning model. The formula is used to update attributes in a backward pass, followed by calculating the gradient ∇ with backpropagation. Formally, the equation is introduced as follows:

$$\theta = \theta - \eta \cdot \overbrace{\nabla_\theta J(\theta;\, x,\, y)}^{\text{Backpropagation}} \tag{4.4}$$

where θ is the weights, biases, and activations in the neural network. ∇ is the learning rate of the model, J is the loss function that we use and in $J(\theta; x, y)$, x is the input and y is the corresponding label for that input, where we feed θ together with the training instances. This optimizer is used in our Vanilla transformer model.

4.6.2 AdamW

Adaptive Moment Estimation (Adam) [13] uses Momentum and Adaptive Learning Rates to converge faster. The optimizer is intended to be suitable for non-stationary objectives and challenges with gradients that are extremely noisy and/or sparse. The method is particularly efficient while dealing with a complex problem containing a lot of data or parameters. AdamW [21] is a variation of the Adam optimizer where it takes advantage of a modified and improved implementation of weight decay (or L_2 regularization) by decoupling weight decay from the gradient update.

Chapter 5

Experimental Evaluation

In this chapter, we first introduce the datasets we use in our evaluations. We perform our experiments on the datasets that have both type tags and schema tags corresponding to each token in the natural language query. We then continue by elaborating on our experimental setup for each model employed in the experiments. Experimental evaluation results of the proposed strategies are provided as compared to state-of-the-art approaches chosen from the literature.

5.1 Datasets

In our experiments, we use three different datasets to evaluate the efficiency and effectiveness of our model and compare it with baselines. We evaluate our models on IMDB, scholar [16], and Yelp [24] datasets which are commonly used in NLIDB systems. The properties of these datasets are provided in Table 5.1. Entity tables are references to the main tables such as 'movie', 'author', and 'business', while transition tables storing the relations among the entity tables, like 'directed-by', 'cast', and 'writes', refer to the relation tables. The number of the columns is presented with 'total attributes', and 'non(PK-FK) attributes' as the title suggests is for the attributes that are neither primary key nor foreign

key. 'total tags' illustrates the total number of the schema tags which is the summation of the non(PK-FK) attributes and total tables.

Table 5.1: Statistics of IMDB, scholar, and Yelp datasets

Properties (#)	**imdb**	**scholar**	**Yelp**
entity tables	6	7	2
relation tables	11	5	5
total tables	17	12	7
total attributes	55	28	38
non(PK-FK) attributes	14	7	16
total tags	31	19	20
queries	131	599	128
tokens in queries	1250	4483	1234

5.2 Experimental Setup

Evaluation of each method is based on the *SacreBLEU* [26] and accuracy using seqeval [23] framework in python. This method is the most common technique for evaluating the models in NLIDB systems. Additionally, we provide n-gram precisions and fmeasures graphs. We split the datasets as 65% for the train set and 17.5% for the validation set and 17.5% for the test set. Our models are trained in Google's *Colab Pro* [4] environment with 16GB GPU and 32GB RAM. The average inference time for an input instance is 2.40 ms which is relatively fast compared to the models that use a lookup table to search for the relevant schema components.

Seqeval is among the most popular frameworks to evaluate sequence labeling evaluation tasks. We use this tool to calculate the accuracy of the generated translations. To calculate the accuracy, we pass predictions and references to seqeval to compute an accuracy score for our generated SQL queries.

SacreBLUE is a metric which assigns a numeric score to the model's translation to evaluate how good the translation is compared to the reference translations. The approach that SacreBLEU and also BLEU [25] take is to compare the precision

of n-grams of the generated translation to the precision of n-grams of the references. In this metric, the number of times that a word appears in the translation is clipped based on the number of times it appears in the reference translation. To deal with the word ordering problems, the geometric mean of several different n-grams is calculated. The difference between the length of the generated translation and the reference translation is also important. The brevity penalty is related to this difference. BLEU has some problems including the fact that it does not consider semantics and it struggles a lot with non-English languages. Another problem with BLEU is that it assumes that human translations have already been tokenized and this makes it hard to compare scores with different tokenizers.

SacreBLEU on the other hand addresses the tokenizer limitations of the BLEU where the score is calculated by passing the whole text and the tokenization is done in the metric.

Our models are trained in Google's *Colab Pro* [4] with 16GB GPU and 32GB RAM.

We train the model to find the optimal hyperparameters by trying different optimizers, loss functions, dropout values, learning rates, etc. As mentioned in the methodology chapter, the optimum loss function for most of the language models is cross-entropy loss and it is the same for both Vanilla and T5 models. Furthermore, we found different optimizers that work better in each model. We use SGD on the Vanilla and AdamW on the T5-base model. Table 5.2 summarizes the optimal hyperparameters for each model.

Table 5.2: Optimal hyperparameters for our models

Model	**Optimizer**	**Loss Function**	**Learning Rate**	**# Epochs**	**batch size**
Vanilla Transformer	SGD	cross-entropy	5x10-3	1000	8
Pre-Trained T5	AdamW	cross-entropy	5x10-5	30	8

5.3 Evaluation Results

In this section, we provide the performance evaluation results to compare our methods with their variants and also the other approaches. The experiments have been conducted with the list of parameters presented in Table 5.2. We compare our models with NALIR [17], TEMPLAR [3], DBTagger [37], and also not-enriched versions of our models. Table 5.3 provides the performance results in terms of accuracy obtained with three different datasets. As shown in this table, our model outperforms other existing methods in all three datasets.

Table 5.3: Comparing the translation accuracy of the TranSQLate methods with state-of-the-art translation solutions on different datasets

model	**imdb**	**scholar**	**Yelp**
NALIR	0.383	0.330	0.472
TEMPLAR (on NALIR)	0.500	0.402	0.528
DBTagger	0.564	0.551	0.461
Vanilla	0.561	0.528	0.579
Vanilla+TranSQLate	0.601	0.549	0.613
T5	0.742	0.583	0.689
T5+TranSQLate (v_1)	0.736	0.728	0.739
T5+TranSQLate (v_2)	0.748	0.750	0.749

The performance results illustrate the effect of feeding the enriched natural language query to the model. We successfully achieve higher translation accuracy with each of our five model variants on any of the datasets. The most significant improvement is reached on the scholar dataset where we capture a 16.7% boost in the translation accuracy while comparing the T5 with the T5+TranSQLate (V_2). On the other hand with our Vanilla model, we accomplish a 4% boost. Analyzing the results suggests that the second enrichment method proposed on the T5 model provides both the highest overall results and improvements in translation accuracy.

We also provide SacreBLEU and n-gram precision to compare the quality of translation. SacreBLEU is one of the most popular metrics to evaluate the results of the neural machine translation models. Tables 5.4, 5.5, and 5.6 illustrate the improvement in the results.

Table 5.4: SacreBLEU, n-gram precisions, and brevity penalty on IMDB dataset

metric	**T5**	**T5 + TranSQLate (v_1)**	**T5 + TranSQLate (v_2)**
SacreBLEU	63.857	57.347	60.520
unigram precision	79.027	84.034	90.279
bi-gram precision	72.755	81.913	87.854
3-gram precision	67.981	78.280	85.106
4-gram precision	63.183	72.936	81.956
brevity penalty	0.888	0.680	0.701

Table 5.5: SacreBLEU, n-gram precisions, and brevity penalty on scholar dataset

metric	**T5**	**T5 + TranSQLate (v_1)**	**T5 + TranSQLate (v_2)**
SacreBLEU	58.942	62.164	65.490
unigram precision	81.604	78.062	79.679
bi-gram precision	75.885	69.178	72.165
3-gram precision	71.137	62.054	67.781
4-gram precision	66.533	55.348	60.352
brevity penalty	0.801	0.921	0.982

Table 5.6: SacreBLEU, n-gram precisions, and brevity penalty on Yelp dataset

metric	**T5**	**T5 + TranSQLate (v_1)**	**T5 + TranSQLate (v_2)**
SacreBLEU	63.754	64.964	67.734
unigram precision	76.295	70.407	72.847
bi-gram precision	66.966	66.585	69.118
3-gram precision	60.479	63.225	66.160
4-gram precision	53.467	60.091	63.187
brevity penalty	1.0	1.0	1.0

As illustrated in Table 5.4, IMDB is the only dataset that our models could not achieve a higher SacreBLEU score. According to our analysis, different factors impact this score. The first one is the fact that there is a gap between the length of the generated SQL and the reference SQL causing the brevity penalty to decrease. Second, the IMDB dataset seems to be more complex than the other two datasets including more total tags and total attributes as mentioned in Table 5.1. Third, the IMDB dataset has some ambiguity, for instance, the name *Matt Damon* can refer to *producer*, *actor*, and *writer* in different natural language queries which makes it hard for the model to make the right predictions. Finally, since we are obtaining these tags from DBTagger, we need it to perform well on any dataset but according to our analysis, DBTagger's worst performance is on the IMDB dataset. Nevertheless, we still observe significant improvement in different n-gram precisions up to 18 points in 4-gram precision. Considering the overall performance, the T5+TranSQLate v_2 model achieves the highest scores in most of the setups.

On the scholar and the Yelp datasets, we achieve 6.5 and 4 more SacreBLEU points while enriching the input. The brevity penalty increases in the scholar dataset with the enriching methods while it does not change in the Yelp dataset while achieving the highest score.

We also include some graphs of the validation set to illustrate the training steps and the convergence of our T5+TranSQLate v_2 model. Figures 5.1 through 5.4 illustrate plots for fmeasure, precision, recall, and loss for IMDB, scholar, and Yelp datasets. Since the number of data instances in IMDB and yelp datasets are very close, their results are similar.

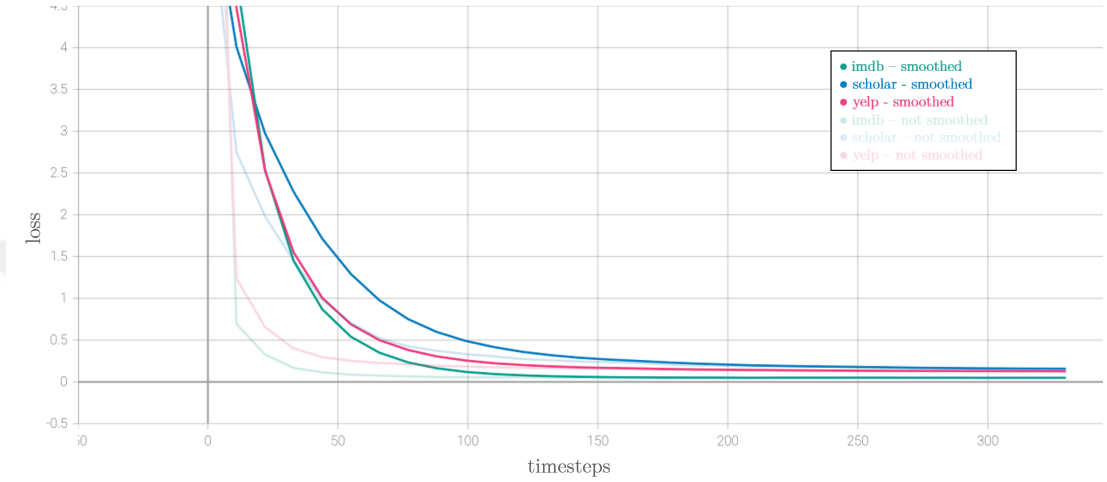

Figure 5.1: loss in the T5+TranSQLate v_2 model on IMDB, scholar, and Yelp datasets with respect to time steps for the validation set

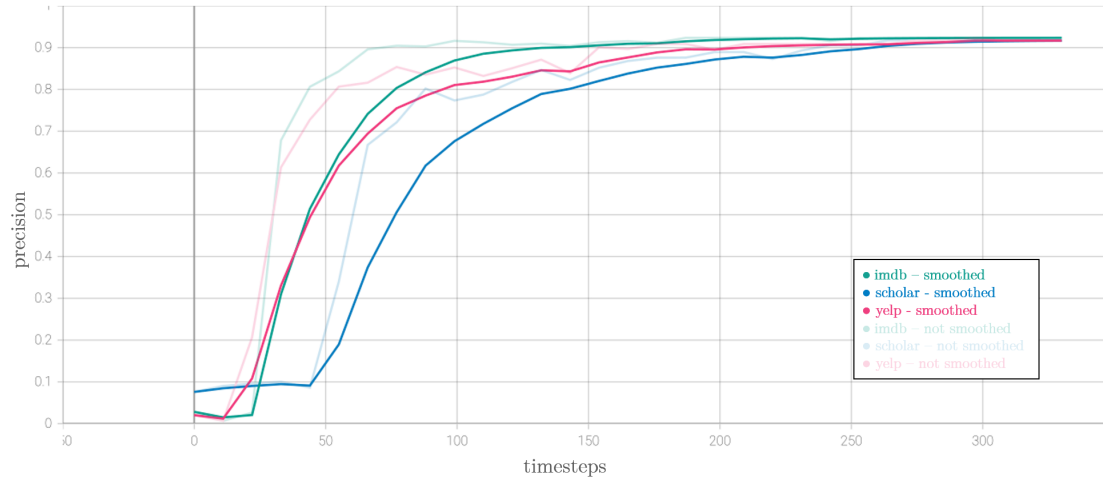

Figure 5.2: precision in the T5+TranSQLate v_2 model on IMDB, scholar, and Yelp datasets with respect to time steps for the validation set

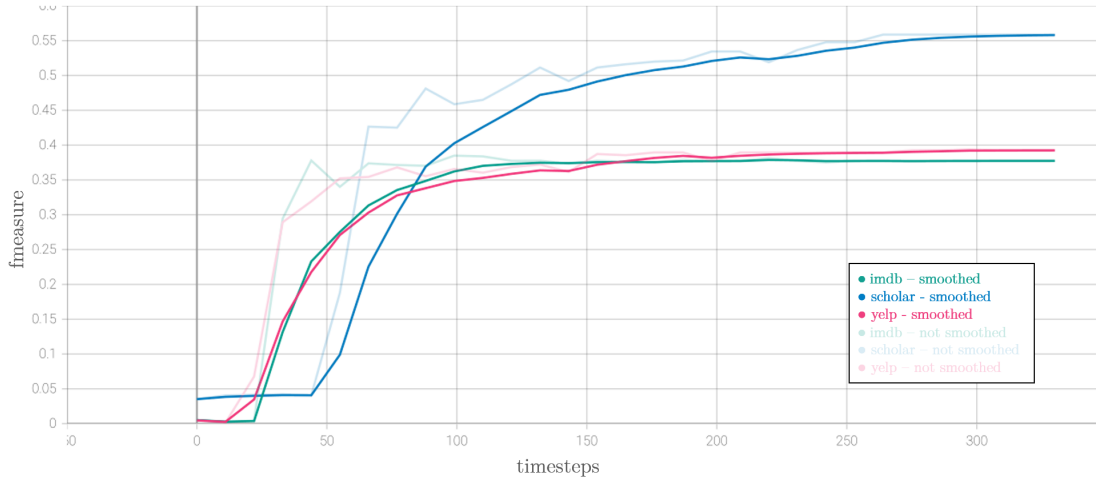

Figure 5.3: f-measure in the T5+TranSQLate v_2 model on IMDB, scholar, and Yelp datasets with respect to time steps for the validation set

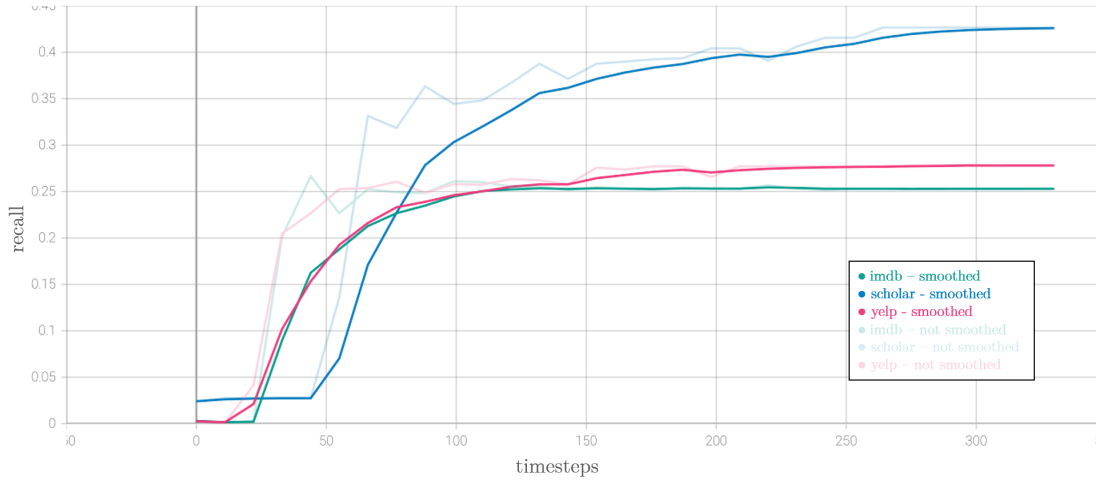

Figure 5.4: recall in the T5+TranSQLate v_2 model on IMDB, scholar, and Yelp datasets with respect to time steps for the validation set

Chapter 6

Conclusion and Future Work

In this thesis, we present three novel input enrichment methods over two transformer-based models to enable more accurate translation in NLIDB systems. The methods are applicable over huge pre-trained models and the models on a smaller scale. We utilize these enrichment methods over the Vanilla transformer by concatenating the embeddings with type tags and schema tags to feed extra information about the database schema to the network allowing it to make better predictions.

Furthermore, in the pre-trained T5 model we feed this information along with the input sequence in two ways and we observe up to 18 points of improvement in n-gram precision, 16.7% improvement in translation accuracy, and 6.5 points in SacreBLEU score. We train our models on IMDB, Yelp [24], and scholar [16] datasets which are heavily used in the NLIDB community and we outperform the state-of-the-art NALIR [17], TEMPLAR [3], and DBTagger [37].

As a future work, we plan to gather type tags and schema tags for some other datasets and show the improvement over these datasets as well. Furthermore, we plan to add a schema linking mechanism to our model by creating a table, attribute, and value lists and searching for each token to check if contains any information about the schema in order to make it independent from tags. As an

extension, we are planning to apply the back-translation method to our model allowing it to translate SQL to English. Finally, we aim to enrich the pre-trained model with additional input layers to compare with other proposed methods.

Bibliography

[1] Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81, 1995.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Christopher Baik, Hosagrahar V Jagadish, and Yunyao Li. Bridging the semantic gap with sql query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 374–385. IEEE, 2019.

[4] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses.

In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006. European Language Resources Association (ELRA).

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R Woodward, John Drake, and Qiaofu Zhang. Natural sql: making sql easier to infer from natural language specifications. *arXiv preprint arXiv:2109.05153*, 2021.

[10] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[11] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

[12] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, April 2017.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*, 2018.

[15] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[16] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, September 2014.

[17] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84, sep 2014.

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

[19] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online, November 2020. Association for Computational Linguistics.

[20] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[22] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[23] Hiroki Nakayama. seqeval: A python framework for sequence labeling evaluation, 2018. Software available from https://github.com/chakki-works/seqeval.

[24] Isil Dillig Navid Yaghmazadeh, Yuepeng Wang and Thomas Dillig. Sqlizer: Query synthesis from natural language. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM*, pages 63:1–63:26, October 2017.

[25] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[26] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, October 2018. Association for Computational Linguistics.

[27] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[30] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[31] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Özcan. Athena: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment*, 9(12):1209–1220, 2016.

[32] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.

[33] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long*

Papers), pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[34] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13806–13814, 2021.

[35] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

[36] Arif Usta. *Towards deeply intelligent interfaces in relational databases*. PhD thesis, Bilkent University, 2021.

[37] Arif Usta, Akifhan Karakayali, and Özgür Ulusoy. Dbtagger: Multi-task learning for keyword mapping in nlidbs using bi-directional recurrent neural networks. *Proc. VLDB Endow.*, 14(5):813–821, jan 2021.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*, 2019.

[40] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[41] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–26, 2017.

[42] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*, 2018.

[43] Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. Editing-based sql query generation for cross-domain context-dependent questions. *arXiv preprint arXiv:1909.00786*, 2019.

[44] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.