Master Thesis

Answering questions about insurance supervision with a Neural Machine Translator

J. Glowienke

Master Thesis DKE-21-22

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Data Science for Decision Making
at the Department of Data Science and Knowledge Engineering
of the Maastricht University

Thesis Committee:

Dr. J. Niehues, Dr. G. Spanakis
External Supervisor: Dr. W.J. Willemse (De Nederlandsche Bank)

Maastricht University
Faculty of Science and Engineering
Department of Data Science and Knowledge Engineering

July 9, 2021

Abstract

In this research, we build upon existing work of applying state-of-the-art Neural Machine Translation (NMT) models to translate questions in natural language to graph database queries based on the SPARQL query language. This approach enables laymen to easily retain information from a dataset in a linked data Resource Description Format (RDF), which can be part of a chatbot or conversational system. Existing work exhibits shortcomings for the correct translation of names and other entity objects, which is crucial for the performance measured in terms of getting the correct results using the query. Entity objects are only present occasionally during training or are fully absent and consequently not part of the vocabulary. To improve the performance when dealing with unknown and rare words, we use different approaches from the literature based on subwords and copy methods. We measure the match of names between reference and translation and observe accuracies of up to 91,62% for names, which are not part of the vocabulary of the model. The overall quality of the translation achieves Bilingual Evaluation Understudy (BLEU) scores of up to 94,11 on questions, which were not seen by the model during training.[1]

[1]The code for this research might be available in the future at `https://github.com/DeNederlandscheBank/nqm`

Contents

1 Introduction **6**

2 Preliminaries **8**
- 2.1 RDF and SPARQL . . . 8
- 2.2 Neural Machine Translation . . . 9
  - 2.2.1 Encoder - Decoder . . . 10
  - 2.2.2 NMT with Neural Networks . . . 11
  - 2.2.3 Attention . . . 12
  - 2.2.4 Transformers . . . 14
  - 2.2.5 Generative Pre-training . . . 17
- 2.3 Handling of Rare and Unknown Words . . . 17
  - 2.3.1 Subwords . . . 18
  - 2.3.2 Copying with alignments . . . 18
  - 2.3.3 Copy and Pointer Models . . . 19
- 2.4 Evaluation using BLEU Score . . . 20

3 Related Work **22**

4 Proposed Approach **25**
- 4.1 Data Generation . . . 25
- 4.2 Tokenization and SPARQL encoding . . . 30
- 4.3 Models . . . 31
  - 4.3.1 Transformer Model . . . 32
  - 4.3.2 Replacing the Out-of-Vocabulary Words . . . 32
  - 4.3.3 Transformer and Aligning . . . 33
  - 4.3.4 Subword Model . . . 33
  - 4.3.5 Pointer-Generator Model . . . 33
  - 4.3.6 XLMR Model . . . 37
- 4.4 Evaluation Metrics . . . 38
  - 4.4.1 BLEU score . . . 38
  - 4.4.2 String Similarity . . . 38
  - 4.4.3 Name Match Accuracy . . . 39
  - 4.4.4 Query Result Accuracy . . . 40

5 Experimental Setups **41**
5.1 Framework 41
5.2 Datasets 41
5.3 Model Setups 43

6 Results & Discussion **44**
6.1 Model A and B 46
6.2 Model C, C2, D and D2 47
6.3 Model E and F 48
6.4 Model G and H 50
6.5 Model I, J, K1 and K2 51
6.6 Model L 53

7 Conclusion & Further Work **54**

Acronyms

BLEU Bilingual Language Understudy.

BPE Byte Pair Encoding.

CNN Convolutional Neural Networks.

DNN Deep Neural Networks.

EIOPA European Insurance and Occupational Pensions Authority.

GLEIF Global Legal Entity Identifier Foundation.

GNMT Google Neural Machine Translation.

GRU Gated Recurrent Unit.

KB Knowledge Base.

LSTM Long Short Term Memory.

MLP multi-layer perceptron.

MT Machine Translation.

NLP Natural Language Processing.

NMT Neural Machine Translation.

OOV Out-Of-Vocabulary.

Ptr-Net Pointer Network.

RDF Resource Description Format.

RNN	Recurrent Neural Network.
SED	String Edit Distance.
SMT	Statistical Machine Translation.
STS	String Similarity.
URI	Uniform Resource Identifier.

Chapter 1

Introduction

Question answering systems are a great asset as they enable laymen to ask questions in natural language and get an answer. Basically, these systems work like a regular search engine, but are restricted to a specific area of knowledge, which they were trained on. Most question answering systems in Natural Language Processing (NLP) are trained on a large amount of text and are then able to answer questions about the text they "read". In this research, we try a different approach as the our knowledge is contained in a graph style format Knowledge Base (KB). We use the approach proposed by Soru et al. [24] and implement the question answering systems by translating questions in natural language to the query language SPARQL. A simple example asking for the location of an insurance company is given in Table 1.1.

Question	In what city is menzis n.v. located?
Translation	SELECT DISTINCT ?o WHERE { ?e eiopa-base:hasIdentifyingName "menzis n.v." . ?e gleif-Base:identifies ?s . ?s gleif-l1:hasHeadquartersAddress ?a . ?a gleif-base:hasCity ?o . }

Table 1.1: Question about Menzis N.V. and the corresponding query

In this setup, SPARQL is treated as foreign language and a Machine Translation (MT) system is used. The translated queries are used to retain the required answers from the KB. Hence, our research problem is to correctly translate questions to the query language and obtain good accuracy for our results. The relevance of our research is that this setup enables to build a conversational AI system, which can answer simple questions about information in the KB. The degree of difficulty in question style depends on the training data. The system could for example be implemented into a chatbot, which makes it attractive to use for rather non-technical users.

For the problem of building this question answering system usign Neural Machine Translation (NMT), we define three research questions:

- To what extent is the approach of the existing work suitable for a different KB and to what extent able to produce syntactically and semantically correct queries?
- How to properly handle the generation of unknown and rare names and id codes in the translation process?
- Is it more beneficial to train the models on known and unknown names or treat all names as unknown during training? Known or unknown means that the names or id codes are part of the vocabulary of the system or not.

As this exhibits a broad research area, we have to introduce several restrictions. The research is limited to the query language SPARQL and a KB in the Resource Description Format (RDF). Also we restrict us to queries used to return a specific answer from the database. We expect that these type of queries are most relevant to possible users of the system. Additionally, we exclusively use NMT to solve the translation task based on Transformer models, as these models are regarded the state-of-the-art.

First, we explain some preliminary knowledge on MT in Chapter2 and lay out more specifically related work on translating question from natural lanuage to SPARQL in chapter 3. Next, we explain how we generate the training data and what model architectures are used in Chapter 4. Lastly, we explain our evaluation metrics to judge the performance of our models. Before we discuss our results in Chapter 4, we explain what experimental setups are used to train the models in Chapter 6. At last, we draw conclusions about the research done and explain what further research possibilities exist, all this in Chapter 7.

Chapter 2

Preliminaries

In this chapter, we explain certain concepts needed to be able to understand the research performed and the models used. We start with RDF and SPARQL, continue on NMT, explain methods to handle rare and unknown words and finish with a short section on the Bilingual Language Understudy (BLEU) score.

2.1 RDF and SPARQL

RDF is designed to present information, which is modelled as graph [8]. If data is modelled as graph, it may be seen as nodes and edges. Nodes are either subject or objects, while the edges are called predicates. A subject node is given some qualities by connecting it to a object. Figure 2.1[1] shows information about

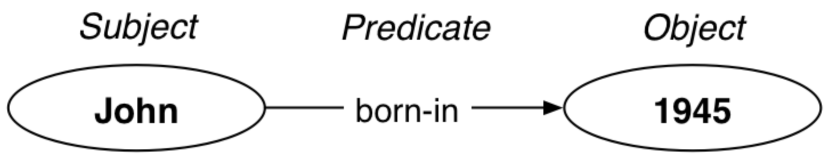

Figure 2.1: Graph Triple

a person called John, which is handled as subject. We then store information about John, which is his year of birth. This is a object of value 1945, which

[1]`https://www.marklogic.com/blog/making-new-connections-ml-semantics/`

is connected to John by the predicate 'born-in'. In a tabular data format, this might be modelled by John being a row, the year of birth a column and the value a cell at the intersection of the respective column and row. RDF is used to serialize data stored as graph and is essential to the Linked Data Web, which is also called the Semantic Web. Uniform Resource Identifier (URI) are used to identify information in the graph database. A URI is similar to an URL, but does not need to follow a general format and can be more user-specific. Normally, a URI consists of a prefix and the entity defined by it, where the prefix defines the namespace of the database. This namespace indicates where the information is to be located.

SPARQL is a structured and syntactically defined language that enables user to query a RDF graph by specifying templates to be satisfied by the data. Usually templates specify a triplet consisting of subject, predicate and object. The query will then return data from the graph, which matches the required triplets. In our research, we are mainly concerned with 'select' queries, but more is possible in SPARQL.

Listing 2.1: SPARQL Query Example

```
SELECT DISTINCT ?a WHERE {
?e eiopa-Base:hasIdentifyingName "achmea zorgverzekering n.v.".
?e eiopa-Base:hasInsuranceUndertakingID ?a .}
```

In the example of a SPARQL query in Listing 2.1, we define two triplets the data should match. First, we ask for the subject which has name "Achmea Zorgverzekeringen N.V." and then we ask for the insurance ID code of this subject. These two triplets are surrounded by a "select" statement stating that we want variable "a" returned, which is placed as object of the predicate concerning the insurance ID code. The URI of our database a not human readable and consist of complicated number and letter sequences and therefore querying on these is rather difficult and we use two triplets to prevent using the subject URI directly.

2.2 Neural Machine Translation

MT is a classic sub-field of NLP, which investigates how to translate one language to another without human involvement. This largely involves the general task of NLP to let computers understand language at a semantic level [31]. The conventional methods for MT included rule-base, phrase-based and statistical translation systems, but the research advances on artificial intelligence and Deep Neural Networks (DNN) lead to the proposal of NMT systems. The reason was the good performance of DNN for other NLP tasks and unresolved problems in MT. Traditional MT systems rely heavily on hand-crafted features based on linguistic intuition and creating these features is a tedious process [31]. DNN show exceptional performance for other NLP tasks and are a promising approach to MT. In the following, we highlight the most relevant developments in NMT leading to state-of-the-art Transformer models, where we limit our discussion on vanilla Recurrent Neural Network (RNN) based NMT to prevent confusion by many possible small adaptations and focus on the trace of NMT development.

NMT is designed as an end-to-end learning task. The process of generating a target sequence from a source sequence is also named sequence-to-sequence learning (seq2seq). seq2seq learning can be interpreted as a high-dimensional mapping problem, where the model tries to correctly map the the input and output sequence in the semantic space [31]. One challenge of seq2seq learning is the inherent different length of input and output sequences. Therefore, most modern NMT models encode the input sequence to a fixed-length representation and uses this encoder to generate the output sequence in the decoder. From a probability point of view, the whole task of NMT can be described Equation 2.1.

$$\text{argmax } P(T|S) \tag{2.1}$$

A NMT model tries to generate the target sequence T in order to maximize the conditional probability given the source sequence S.

2.2.1 Encoder - Decoder

As explained above, DNN were already used in various NLP tasks before being applied to NMT. Cho et al. combined a Statistical Machine Translation (SMT) system with a neural network model [2]. The authors introduced the encoder-decoder structure still widely used by most NMT models. The proposed architecture consists out of an encoder and decoder, both being a RNN. "The encoder maps a variable length source sequence to a fixed-length vector and the decoder maps the vector representation back to a variable-length target sequence" [2].

The RNN is a generalization of a feed-forward neural network to sequences [26]. It is a neural network consisting out of a hidden state h and an optional output y operating on a variable length sequence $x = (x_1, \dots, x_T)$. For each time step t, the RNN updates the hidden state h_t by:

$$h_t = f(h_{t-1}, x_t). \tag{2.2}$$

f is a non-linear activation function, which may be a simple elementwise sigmoid function or a more complex Long Short Term Memory (LSTM) unit [12]. Cho et al. actually propose a new hidden unit called Gated Recurrent Unit (GRU). The more complex hidden units form a possible solution to the vanishing gradient problem in RNN. The RNN is trained on predicting the next element of a sequence and by this, learns a probability distribution over the sequence:

$$p(x) = \prod_{t=1}^{T} p(x_t|x_{t-1}, \dots, x_1). \tag{2.3}$$

The RNN encoder-decoder architecture of Cho et al. is learning the conditional distribution of a variable input sequence over a variable length output sequence $P(y_1, \dots, y_{T'}|x_1, \dots, x_T)$. The encoder RNN reads in a sequence x one-by-one and updates the hidden states, which after completion of the input sequence represents a summary c of the input. The decoder RNN works slightly different to the vanilla RNN described before. It tries to predict the next element of the output sequence y_t using the previous output y_{t-1}, previous hidden state h_t and the fixed-length representation c of the whole input sequence [2]. The conditional probability of the next symbol is

$$P(y_t|y_{t-1}, y_{t-2}, \dots, y_1, c) = g(h_t, y_{t-1}, c), \tag{2.4}$$

where g is a activation function, which produces valid probabilities summing to 1. Using Equation 2.4, we can clearly observe the next symbol of the output sequence y_t being dependent on the previous output, the hidden state of the decoder at t, which summarizes the context of the output so far and the summary of the input sequence c. This approach solves the problem of mapping a variable length sequence to another variable length sequence. Cho et al. used 1000 hidden units of their newly proposed GRU and learned embeddings of dimension 100 for each word. The number of hidden units defines the size of the fixed-length representation of the input, meaning that each sequence here is mapped to a semantic representation of size 1000. The authors provided the basis for pure NMT models, though a main part of their model is still SMT. The decoder-encoder structure used for SMT improves overall translation performance in terms of BLEU scores, but exhibits shortcomings in understanding and producing longer sequences. The qualitative analysis shows that the model is able to produce semantically and syntactically representations of phrases [2].

2.2.2 NMT with Neural Networks

The success of the model of Cho et al. inspired new approaches to NMT. Sutskever et al. propose a model for NMT based on deep RNN and making almost no assumption on model structure [26]. The results of this model are comparable to SMT systems and make NMT models based on a RNN architecture the de facto model for NMT.

The authors used a similar approach for sequence learning like Cho et al. and use one RNN as encoder to map the input sequence to a fixed-length representation vector and then map the representation to a variable length output sequence. In order to enable the RNN to deal with long-range temporal dependencies, Sutskever et al. use LSTM units [12]. The goal of the LSTM is to estimate the following equation:

$$P(y_1, \dots, y_{T'}|x_1, \dots, x_T) = \prod_{t=1}^{T'} P(y_t)|v, y_1, \dots, y_{t-1}) \tag{2.5}$$

The LSTM first computes the fixed-length representation v of the input x given by the last hidden state of the RNN and then calculating the probability of the output sequence using the same formulation as Cho et al. By using the fixed-size representation, the problem of different lengths in input and output is solved. In the concrete implementation, there were 3 differences present. The authors used two separate deep LSTM with 4 layers as encoder and decoder. Using two separate networks comes at no computational cost, but makes it possible to train on multiple languages [26]. Deep LSTM clearly outperformed shallower network. Lastly, the word order of the input sequence was reversed. In natural language, words with the same meaning often stand roughly the same position in a sentence. Sutskever at al. state that reversing the word order creates more short-term dependencies between input and output and improved performance of the LSTM [26]. One difference to Cho et al. is that no separate methods from SMT are used to generate the output, but only the probabilities produced by the networks and a right-to-left beam search.

The results reported by Sutskever et al. show drastic improvements to SMT systems. This is impressive seen that little optimizations had been performed and the model is quite simple and straightforward [26]. Traditional SMT systems consist out of many subcomponents tuned separately, while the model of Sutskever et al . only consists of one big neural network. The vocabulary of the pure NMT model was limited and no assumptions about the problem structure were made. SMT usually operate on large or even unlimited vocabularies and require domain assumptions. Moreover, the model encountered no problems correctly translating long sentences, which had been the case for other model using DNN for MT [2].

2.2.3 Attention

As set forth in Section 2.2.2, NMT shows great performance and great potential for further improvements. Bahdanau et al. were concerned with the shortcomings of NMT with longer sequences. According to them, the compression of the complete input sequence into one fixed-size length vector by the encoder, puts too much pressure on the system and might lead to a bottleneck for performance [1]. The authors propose to extend the current encoder-decode architecture by enabling the model to focus on parts of the source sentence, which are relevant for predicting the next target word, called RNNSearch. This concept is named attention and qualitative analysis show that the alignments found by the model align wel with human intuition. Each time the model generates a target word, it searches for the most relevant information concentrated in a set of source positions. Instead of encoding the complete input sequence into one single vector, the newly proposed model architecture encodes the input sequence into a sequence of vectors and chooses a subset of these vectors adaptively to the decoding process [1]. The proposed model is more robust to length of the input sequence and outperforms the encoder-decoder model of Sutskever et al.

on a English-to-French translation task.

For the encoder of RNNSearch, the author use a bi-directional RNN. When using a uni-directional RNN the hidden states only summarize the current word read in and the previous words, as the input is read from the first x_1 to the last x_T. Bahdanau et al. think it is more useful, if the annotation of a word in the input sequence reflects the context before and after in the sequence. The bi-directional RNN use a forward and backward RNN to calculate a set of forward hidden states $(\overrightarrow{h_1}, \ldots, \overrightarrow{h_T})$ and backward hidden states $(\overleftarrow{h_1}, \ldots, \overleftarrow{h_T})$ which are concatenated to one hidden state. Accordingly the hidden state $h_j = \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right]$ of word j contains information about the preceding and following words [1].

The decoder of the RNNSearch model has more changes compared to the model of [26]. The conditional probability for a word in the target sequence defined in Equation 2.4 is changed to:

$$P(y_t|y_{t-1}, y_{t-2}, \ldots, y_1, x) = g(s_t, y_{t-1}, c_t), \tag{2.6}$$

where s_t is the RNN hidden state at time t calculated using:

$$s_t = f(s_{t-1}, y_{t-1}, c_t). \tag{2.7}$$

f represents the non-linear function used inside RNN, which is not closer defined in the original publication, but is likely an LSTM. The import change to note in Equation 2.6 is that instead of using only one context vector c, a specific context vector c_t for each target word y_t is used. The context vector c_t is calculated using the annotations $(h_1, \ldots, h_T)$ produced by the encoder (compare above). These annotations contain information about the complete input sequence with one annotation h_i focussing on the surroundings of word i [1]. Using

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j \tag{2.8}$$

we calculate the context vector for target word t by using the weighted sum of the annotations weights αt_j and the annotations h_j over the whole input sequence. The weight α_{tj} is calculate using:

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{ik})}, \tag{2.9}$$

where

$$e_{tj} = a(s_{t-1}, h_j) \tag{2.10}$$

is an alignment score, explaining how well the inputs around position j and outputs around position t match [1]. This score is based on the hidden state s_{t-1} of the decoder before being concerned with y_t and annotation h_j of the input sequence. By using a softmax like expression to calculate the annotation weights in Equation 2.9, we ensure that the weights sum to 1. The alignment model a

is a feed-forward neural network and is parametrized as part of the model and hence it is trained jointly with the other elements of the model. The weight α_{tj} reflects the importance of annotation h_j with respect to the information of the source sequence up to t stored by hidden state s_{t-1} in deciding on the next hidden state s_t and output y_t [1]. In fact, the decoder can decide which part of the source sequence to pay attention to when generating the output sequence. It is not necessary anymore to encode the whole input sequence into one single fixed-length representation vector. It can adaptively decide where to look for relevant information.

Luong et al. [16] improved the approach of Bahdanau et al. by differentiating between local and global attention. The global attention approach resembles the one used by [1] and calculates the attention weights using all hidden states of the input sequence. This approach is computationally expensive, especially for longer sequences, although Luong et al. simplified the computational path of the the original RNNSearch model slightly. The local attention model only attends to a limited number of source words. The proposed approach is to first get the aligned position p_t for the target word at time t and then derive the context vector c_t by focusing on the set of hidden states within $[p_t - D, p_t + D]$, where D is set empirically. Both attention models outperform NMT models without attention and the local attention model shows good performance when replacing unknown words.

The successes of NMT models with attention mechanism led to a first industrial NMT system. Wu et al. propose the Google Neural Machine Translation (GNMT) system [30]. The authors made a drastic improvement to NMT models by using a very-deep LSTM with 8 encoder and decoder layers, which they were able to by enabling parallelization to reduce training time and reduce the computational load. Parallelization is enabled by using residual and attention connections between the encoder and decoder network. The bottom layer of the decoder is connected to the encoder layer of the encoder. Residual connections add the input of a layer element-wise to the output of the layer, which improves the gradient flow and enables stacking more LSTM layers. Furthermore, the authors employ subwords to handle the translation of rare words and implement length normalization to the beam search. The latter helps to create longer target sequences. The GNMT model achieves state-of-the-art results on then WMT14 English-to-French and English-to-German benchmarks [30]. The authors combined many different research results and create a strong model implementation.

2.2.4 Transformers

RNN and Convolutional Neural Networks (CNN) based encoder-decoder architectures are the predominant choice for most NMT systems. Vaswani et al. [28] propose a new network architecture called Transformers. These are solely based on attention mechanism and give raise to superior performance with more possibilities to parallelize and decrease training time. The transformer model still uses the logic of encoder-decoder models to map an input sequence $(x_1, \ldots, x_n)$ to a fixed-length representation $(z_1, \ldots, z_o)$. Given the representation $\mathbf{z}$, the decoder generates an output sequence $(y_1, \ldots, y_m)$ one element at a time in auto-regressive manner [28].

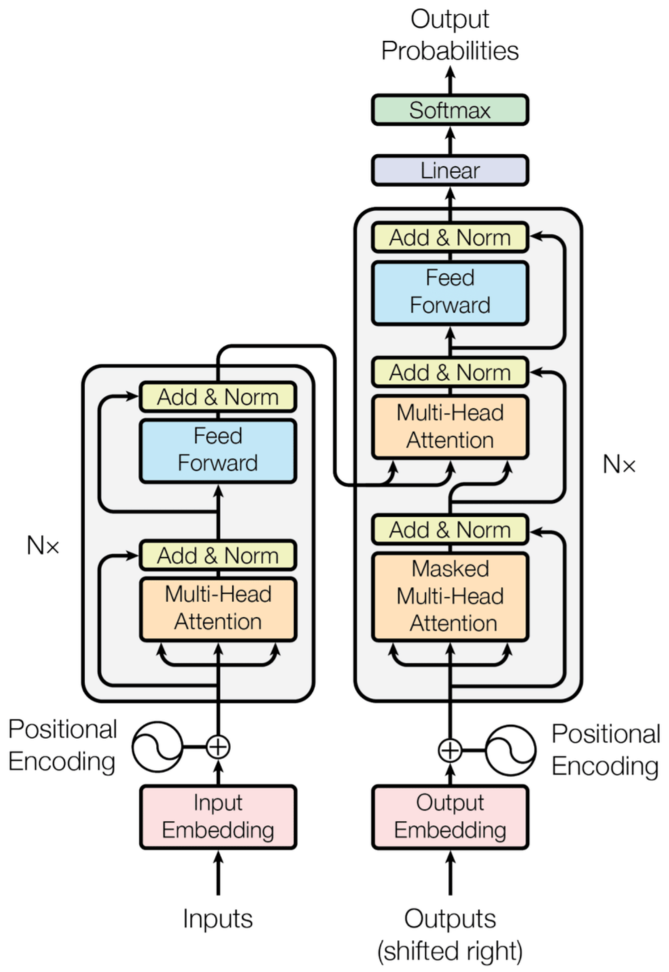

Figure 2.2: Transformer Model Architecture [28]

Vaswani et al. propose a stack of 6 identical layers as encoder and decoder, which schematic is shown in Figure 2.2. Each encoder layer consists out of 2 sub-layers. The first part is a multi-head attention layer and the second is a feed-forward neural network. A residual connection is used around each sublayer and is added to the output of the sublayer, which is consequently normalized.

The output of a sub-layer is hence best described by:

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{2.11}$$

The decoder stack has the same structure, but here another sublayer is added. This sublayer performs multi-head attention over the output of the encoder stack [28]. The same concept for layer normalization and residual connections as in the encoding layers is implemented. The multi-head attention mechanism over the outputs is modified and right-shift to ensure that only known outputs are used, i.e. at position i only outputs before i are used.

The attention mechanism maps a query and set of key-value pairs to an output, where the queries, keys, values and output are vectors [28]. The output is defined as the weighted sum of the values, where the weights are calculated using the compatibility of a query with the corresponding key. This weights indicate the position of values the model should attend to. Depending on the position in the model, the origin of the key, value and query vectors varies, which will be explained later. The "Scaled Dot-Product Attention" uses the queries and keys of dimension d_k and values of dimension d_v. The matrix of outputs is calculated using:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.12}$$

The authors compute the dot products of all keys with the query values, divide this by $\sqrt{d_k}$ and apply the softmax function to obtain the weights correctly scaled. The dot product attention is fast to calculate because of efficient matrix multiplication algorithms and by scaling with $\frac{q}{\sqrt{d_k}}$ the gradients of the softmax will be more stable [28]. In the practical implementation, the authors run 8 parallel attention layers or heads. The parallel attention layers are fed with linear projections of the queries, values and keys. The parallel outputs are concatenated and projected again, resulting in the final output. This multi-head attention allows the model to jointly use information from different positions. The multi-head attention is used in distinct ways. In the encoder-decoder attention, it is used to combine information from the encoder and previous decoder layer, where the queries originate from the previous decoder layer and the keys and values represent the encoder information. Also self-attention is used, where the all three key, value and query vectors come from the same source, e.g. word embeddings, and is able to attend all positions in the previous layer [28].

As the attention mechanism does not store relative information about positions in the source sentence, which is naturally done by RNN, Vaswani et al. feed in positional encodings. These are added to the input embedding of the encoder and decoder stack, as can be seen at the bottom of Figure 2.2. The position in the sequence is encoded using sine and cosine functions.

2.2.5 Generative Pre-training

One challenge for achieving high performance for NLP tasks is that there is a lot of text data generally available, but most of it is not labelled. For NMT tasks a parallel corpus needs to be generated or for tasks like sentiment analysis or language inference concrete classifications are required. Devlin et al . [4] propose the BERT model, which is a deep bidirectional Transformer model for language understanding. The model is pre-trained on an unlabeled language corpus to get a general knowledge of the language and vocabulary. The novelty of this model is the bi-directional approach to achieve better understanding. The model is pre-trained using a masked approach and next sentence prediction. For the masked language task, the model is fed the input sequences with a certain percentages of tokens replaced by a mask token. The task of the language model is then to correctly predict the mask token. In the next sentence prediction, the model needs to predict the adjacent sequence in a corpus, which enables to model to better understand cross-sentence context. A pre-trained BERT model can then be fine-tuned for the specific task by adding an extra layer at the end of the network suitable for the task and train on the desired specific language corpus. The use of pre-trained word and text-pair representation enabled state-of-the-art performance for various tasks with out the need for task-specific architectures.

Due to the success of the BERT model and the open-source availability many adaptations were created and pre-trained models are vastly available. Conneau et al. [3] pre-train a model based on the RoBERTa architecture to create cross-lingual representations. The XLM-R model is pre-trained on a common crawl dataset of 100 languages using the masked multi-lingual language model approach. The results show high performance on various language tasks and also for low-resource languages. Although the model was mainly tested on sentence classification, sequence labelling and question answering, We propose an architecture to make use of this model for our translation tasks in Section 4.3.6.

2.3 Handling of Rare and Unknown Words

Most NLP, and accordingly also NMT, problems typically operate with a vocabulary of fixed size. Limiting the size of the vocabulary reduces memory and computational power. However, language in general is not operating on fixed size vocabulary and there exist many rare words, which are not frequently used. These might not be contained in the limited vocabulary used for a NMT model, unless we grow the vocabulary really large, but then the computational load would grow to extremes. Additionally, there might also be words, which were completely unknown when training a model. This is especially the case when operating with entity names or id codes, as we do in this research. In the following, we describe several strategies how to handle rare or unknown words for

NMT.

2.3.1 Subwords

Sennrich et al. [23] propose to make NMT models handle open vocabulary translation by splitting rare and unknown rare words into subwords. Their idea is based on the notion that certain words, for example compounds, are translatable via smaller units than words. One concrete example the authors give is the german word Abwasserbehandlungsanlage, which translates as sewage water treatment plant. Splitting of such a rare word into subwords and translating it, seems like a more promising approach compared to encoding this long word as one fixed-length vector. Hence, the approach is to split rare words into subwords as preprocessing step and the let the model operate on the subwords. The produced translations also is made up of subwords and is concatenated back to regular words.

As concrete algorithm for generating the subwords, Sennrich et al. use variation of Byte Pair Encoding (BPE) to words. The idea is to start of with a character vocabulary and and represent each word by characters plus an end-of-word token to be able to restore the original words. All symbol pairs are counted the most frequent pair of style ('x','y') is merged to ('xy). This is done a predefined number of times, which is the only parameter of the algorithm. The final vocabulary consists of all characters and the frequent character n-grams, which were merged in the process. Depending on the number of merge operations, a varying amount of frequent words stay as word, while rare or unknown words are split into subwords unit and passed to the NMT model.

The authors show in their work that using subword units improves translation quality by up to 1.4 BLEU points and especially improve the translation unigram performance of rare words in the data set. The better performance is observed by training joint BPE, meaning that BPE are learned simultaneously on source and target. This prevents certain subwords only being observed in the source or the target language. The results prove the capability of NMT to work on open vocabulary problems.

2.3.2 Copying with alignments

Luong et al. [17] propose to counter the problem of rare or unknown by using an alignment-based technique. They implement different approaches, which are based on the concept of using an alignment algorithm to let the NMT model output for each word in the target sequence the corresponding word in the source sequence. Using the alignments, it is then possible to trace back the origin of an Out-Of-Vocabulary (OOV) marker in the target sequence and replace it the corresponding word in the source sequence. The authors use an alignment dictionary or the identity translation of the replacement. The alignment dictionary is created using an unsupervised aligner and contains matching words from the target and source corpus.

Three different models are implemented which are slight variations of one another. The most promising approach is the positional unknown model, where a unknown word in the target sequence is labelled with the relative position to the corresponding word in the source sequence [17]. The logic used by Luong et al. can be applied any type of NMT model and we make use of this and apply it for a transformer model. Our implementation is slightly varied compared to Luong et al. and details can be found in Section 4.3.3.

2.3.3 Copy and Pointer Models

A different method for dealing with unknown and rare words in NLP models is to directly copy or point elements from the input sequence to the output and not generating it from the vocabulary. This idea is based on the observation that in many NLP applications, e.g. summarization, OOV words on the target side are replicates of words in the source sequence [9].

Vinyals et al. introduced a architecture called Pointer Network (Ptr-Net) [29]. The Ptr-Net is using the attention mechanism of [1] integrated in a RNN to generate a output sequence consisting of discrete tokens referring to elements of the input sequence [29]. The authors applied the Ptr-Net to 3 different combinatorial problems and showed enhanced performance compared to the baseline, whilst having variable sized output dictionaries. Consequently, the Ptr-Net was used to create new approaches to NMT (Gulcehre et al. [10]) and summarization (CopyNet, Gu et al. [9]).

Gulcehre et al. incorporate two softmax layers into their RNN with attention model: when predicting the next word in the output, one softmax predicts the location of a word in the source sentence, the other softmax predicts a word from the vocabulary [10]. At each timestep, a multi-layer perceptron (MLP) makes the decision which prediction of the two softmax layers to use. The authors argue that humans tend to reuse the original word, if they do not understand it or do not know the translation. Accordingly, the give their model the same skill by learning how to point and when to point [10]. The MLP uses the representation of the context sequence and previous hidden state of the RNN to calculate a probability on which softmax to use [10]. The second softmax layer described above is called the location softmax and is a pointer network like in Vinyals et al. Gulcehre et al. added the ability to the pointer network to decide whether to copy a word from the input sequence or use the prediction from the target dictionary.

See et al. further developed the idea of pointer networks and propose a pointer-generator network for summarization. Their architecture augments the seq2seq attentional model using RNN "in two orthogonal ways" [22]. They combine the hybrid pointer-generator network to copy words from source text via pointing and use coverage to prevent repetition. Opposing to [10], See et al. mix the probabilities from the copy and vocabulary distribution. The chosen approach is closer to the CopyNet of [9]. A more detailed of the mechanism is given in Section 4.3.5.

The pointer-generator model was used by Enarvi et al. for a summarization task with patient-doctor conservations [6]. The implement the pointer-generator approach of [22] to use it with a transformer architecture [28]. As summarization is also a seq2seq problem, we can employ their architecture for our objective of translating questions to query language. Their experiments show improved performance for the combination of transformer and pointer-generator model compared to RNN, RNN-search and RNN with a pointer-generator architecture models [6]. The authors of [6] kindly included their code to the fairseq framework used for our research and hence we can test the performance of the pointer-generator model on our research task. More details for the implementation are given in Section 4.3.5.

2.4 Evaluation using BLEU Score

The BLEU score was invented by Papineni et al. in 2002 to simplify the evaluation process of machine translations, which was mainly done by human judgement previously. This process was slow and their motivation was to increase the output of innovative research ideas, which were jammed in the "evaluation bottleneck" [20]. The viewpoint and central idea of the authors is that "the closer a machine translation is to a professional human translation, the better it is" [20]. The authors also prove the strength of BLEU scores by the comparing BLEU scores and human evaluation results and concluding that the correlation is high.

The basic implementation of the BLEU score is to compare n-grams of the translation to n-grams of the reference and count the number of matches. A n-gram is a subsequence of a sentence consisting of n words. The number of matches is then divided by the total number of n-grams in the candidate translation. However, Papineni et al. noted that MT systems tend to overgenerate reasonable words, which results in high precision translations, but with unreasonable content [20]. Therefore, they propose to cap the count of n-grams in the translation by the count of the n-grams in the reference, see Equation 2.13. Then, the capped counts are added up and divided by the total uncapped number of candidate translation n-grams.

$$\text{Count}_{\text{clip}} = \min(\text{Count}, \text{Count}_{\text{ref}}) \tag{2.13}$$

These modified n-gram precision scores are calculated for a range of values of n. The value of N_{max} is set as hyperparameter and can be freely chosen in a reasonable manner. The final score is calculated using the geometric mean of the modified n-gram precision scores [20].

As further adaptation, a brevity penalty is added. Papineni et al. noted the modified n-gram precision fails to enforce proper translation length and favours short translations [20]. The penalty is calculated in such a way to punish short translations. By letting c be the length of the candidate translation and r the effective reference length, the brevity penalty is calculated by Equation 2.14.

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r)/c} & \text{if } c \leq r \end{cases} \tag{2.14}$$

The value of the brevity penalty is between 0 and 1 and is 1, if the length of the candidate translation is longer than the reference translation. Combining the brevity penalty and the geometric mean of the modified precision scores, the complete formula for the BLEU score is written in Equation 2.15. The weights w_n should add to 1 and be positive. Experimentally, $w_n = 1/N$ and $N_{\mathrm{max}} = 4$ have shown good performance [20].

$$\mathrm{BLEU} = \mathrm{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \cdot \log p_n\right) \tag{2.15}$$

The authors also state the the BLEU score can be effectively used, if there is only one reference translation available.

Chapter 3

Related Work

The focus of our research has been the translation of questions posed in natural language to the query language SPARQL. Furthermore, we specified that techniques from the field of NMT should be used. There is a good deal of research done using classic Machine Translation techniques and also to other structured languages like SQL. However, we narrow down the related work to the combination of NMT and SPARQL, since sufficient material is present to build a starting point for our own work.

At first, Question Answering system enabled users to build SPARQL queries using natural language, but these systems lacked performance in case of vocabulary mismatch. Soru et al.([24],[25]) are the first to introduce NMT for parsing SPARQL queries. They propose the "Neural SPARQL Machine" for end-to-end or *seq2seq* translation of a NL expression to SPARQL queries. The proposed architecture relies on a *generator*, a *learner* and a *interpreter*. The task of the *generator* is to build a dataset using hand-crafted query templates. The authors of [24] argue that knowledge bases (KB) on the semantic web contain user-specific classes, predicates and instances. Therefore, the dataset needs to match the KB, since certain keywords used in one KB might be of no meaning in a different KB. A more detailed explanation on the generator is given in 4.1. The *learner* module features an encoder-decoder RNN with LSTM units. The learner is used to encode the NL questions into a fixed-length representation. The interpreter at hand is decoding this representation to build the SPARQL query. The authors used DBpedia as KB, focusing on the 'dbo:Monument' class. As basis for the dataset, 38 query templates were used and filled with 300 or 600 instances per template. One hundred examples are taken from the working dataset to build up validation and test set. They achieved a BLEU accuracy of around 0.8. The model is able to grasp the essence of a unseen query at test time, but it is not clear from the paper whether unknown URI are used at test time. In [25], some more work was added and different encodings were tested. The results from this work show that the used encoding for the SPARQL queries (see 4.2) strongly influence the performance of the model. By shortening the sequences and using direct entity translations, the BLEU accuracy was raised above 0.9.

Hartman et al. ([11]) looked further into generating large datasets, which can be used for answering questions on a KB using neural networks. Their work is specific to DBpedia. They introduce the 'DBNQA' dataset, which consists out of 894,499 pairs of questions and SPARQL queries. The data generation process used in [24] is further extended to classes outside 'dbo:monument', and a detailed explanation of the generation steps is given. The scope is to enable the generation of datasets for other KBs, which is especially useful for our work (see 4.1). A reuse of the dataset does not seem useful, since it is specific to the DBpedia KB.

Panchbhai et al. ([19]) add more work to the field of seq2seq models for SPARQL query parsing. The authors investigated compositionality, when translating SPARQL queries. Compositionality is the ability to generalise on the seen examples and be able to compose an unseen example, which is built from two seen examples. An example for this is the following; the questions 'When was Barack Obama born?' and 'Who was the 44th President of the USA' are merged into the question 'When was the 44th President of the USA born?'([19]). They tested this ability for a standard seq2seq model setup with an 80%-10%-10% split for the dataset. The achieved result was 89.75% on accuracy, which was barred from perfect accuracy due to unknown or mismatched entities at test time. Entities are names or objects, which both question and query are concerned with. The authors additionally implemented a more automatized approach for template generation. They used a ranking mechanism based on the popularity of the answer in the KB. For the rank, the depth of the question was also accounted for by using a dampening factor. Moreover, the authors created also a different task to test the abilities of the model. The fixed the vocabulary for the train and test set and ensured by the use of a frequency threshold that the model is able to sufficiently learn words to be tested. Furthermore, the train set would contain more single questions, where the train set contained more questions composed of single questions. The best accuracy was 63% and BLEU score of 93 by the use of attention and pre-trained embeddings. The low accuracy, whilst having a high BLEU score, was mainly due to entity mismatches.

As last relevant paper to be mentioned here, the work of Yin et al. ([32]) is summarized. This paper is based on the work done by the authors above and contributes to the field by testing the performance of 8 different NMT models including RNN, RNN with attention, CNN and Transformers. The models were trained and evaluated on three different datasets, which were also used in [24] and [19]. The performance of the models was done by BLEU scores, perplexity and accuracy of the translations, which was for testing evaluated on a fixed set of 100 examples like in [24]. The further model pipeline concerning data generation and SPARQL encoding was kept in line with [24], [25], [11] and [19] to ensure comparability. The best observed BLEU score was 95.31 on the Monument800 ([24]) and 68.82 for the DBNQA ([11]) using a Transformer model. The authors again observed problems with entity mismatches and noted that erroneous replacement was done across all queries, where consistently the correct entity was replaced with the same erroneous one. Furthermore, some models struggled with the large vocabularies of DBNQA, which is due to the generation process of [11], where a frequency threshold is applied for the entities.

Chapter 4

Proposed Approach

In the following chapter, we wil explain our approach to translating questions in natural language to the query language SPARQL. First the data generation process and encoding process is explained. Then we explain the concrete models used and the evaluation methods. The model section is mainly concerned with the architectures of the models and concrete details on datasets used and training setup are given in Chapter 5.

4.1 Data Generation

In this section, we will describe in detail the process to generate the language dataset. This process is inspired by [24] and [32], whose contributions were described in Chapter 3.

Our work is concerned with data about European insurance companies. We retrieve two datasets, one is a RDF dataset from the Global Legal Entity Identifier Foundation (GLEIF) register [1], the other one is an extract in csv format from the European Insurance and Occupational Pensions Authority (EIOPA) register [2]. Using these two datasets, a new RDF KB in turtle format is built for the data from the EIOPA register. The data obtained from the EIOPA register is in a tabular format and we use a python script to modify this dataset to a RDF graph database. This is done by linking the data in the table to the the predicates listed in Table 4.1 and an example is given in Listing 4.1 for the new way of representing the information of one insurance company. This KB uses the namespace 'eiopa-Base', whilst information from the GLEIF data is rooted in the namespace 'gleif-L1'.

[1]`https://www.gleif.org/en/`

[2]`https://register.eiopa.europa.eu/registers/register-of-insurance-undertakings`

Listing 4.1: Example of EIOPA Graph Object

```
eiopa-L1:IURI-De-Nederlandsche-Bank-L0077
            owl:a eiopa-Base:InsuranceUndertakingRegisterIdentifier ;
    eiopa-Base:hasCrossBorderStatus "Domestic undertaking" ;
    eiopa-Base:hasEUCountryWhereEntityOperates CountryCodes:NL ;
    eiopa-Base:hasIdentifyingName "aegon leven",
        "aegon levensverzekering n.v.",
        "l0077" ;
    eiopa-Base:hasInsuranceUndertakingID "l0077" ;
    eiopa-Base:hasNCA "De Nederlandsche Bank" ;
    eiopa-Base:hasOperationStartDate "01/01/1901 01:00:00" ;
    eiopa-Base:hasRegistrationStartDate "01/01/1901 01:00:00" ;
    gleif-Base:identifies gleif-L1:L-5493003SPEWN841SWG39> .
```

Predicate	Description
hasCrossBorderStatus	Operating in only one country ('Domestic Undertaking') or multiple countries
hasEUCountryWhereEntityOperates	Country insurance is operating in
hasIdentifyingName	Different names used for a company
hasInsuranceUndertakingID	DNB insurance identifier
hasNCA	Supervising authority of the insurance
hasOperationStartDate	Starting date of operation
hasRegistrationStartDate	Starting date of insurance registration

Table 4.1: Predicates in the EIOPA dataset

As the two KBs do not use the same URI as subjects, a link between the two KB using the 'gleif-base:identifies' predicate is created. Hence, in practical use, the two KBs are loaded as one graph object (using the rfdlib python library).

Soru et al. stated in their paper that they "do not expect to reuse a learned model for QA on other KBs" [24]. The concept of Semantic Web allows users to "define their own classes, instances and properties without the need for a global design" [24]. In their RDF KB concerned with data on Monuments from DBpedia, information associated with the location of an entity is stored with the predicate 'dbo:country'. However, this predicate is of no use in our KB described above. Therefore, we have to construct a set of templates to build the language dataset. The desire was to translate all stored information in the KB to a suitable question and query. Additionally, we paid attention to the needs of supervisors at DNB Insurance Supervision and the possible questions on insurance companies they have in their daily work. These questions are mostly about returning some piece of information about an insurance company.

The core of a template is an aligned question and SPARQL query. The query is set up to return the information asked for in the question. An example of a template is given in Table 4.2 and Table 4.3.

Question	In what country does `<A>` operate?
Query	select distinct ?o where {?e eiopa-Base:hasIdentifyingName `<A>`. ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o.}
Generator Query	select distinct ?a where {?x eiopa-Base:hasEUCountryWhereEntityOperates [] . ?x eiopa-Base:hasInsuranceUndertakingID ?a.}

Table 4.2: Insurance ID Template

Question	In what country does `<A>` operate?
Query	select distinct ?o where {?e eiopa-Base:hasIdentifyingName `<A>`. ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o.}
Generator Query	select distinct ?a where{?s gleif-L1:hasLegalName ?a . ?e gleif-Base:identifies ?s . ?e eiopa-Base:hasEUCountryWhereEntityOperates [].}

Table 4.3: Name Template

The query is made fitting to the KB by the use of predicates contained in the KB. For the example in Table 4.2, this is 'eiopa-Base:hasIdentifyingName' and 'eiopa-Base:hasEUCountryWhereEntityOperates'. Additionally, the question and query are made specific by filling in the entities, which the question is mostly about, with data from the KB. For this, the template question and query both contain a placeholder. The placeholder variable is marked by a surrounding less-than- and bigger-than-sign, e.g. `<A>`. It is important for the correct working of the generator module that the variable name of the placeholder and also the other SPARQL variables (e.g. '?a') only contain one letter. Additionally, the variable queried on in the generator query, directly after 'select' or 'select distinct' should match with the placeholder in the query and question.

In order to fill the placeholder variable with data from the KB, a generator query is constructed as part of the template. This generator query is used to extract entity names or insurance IDs from the KB, commonly called entity objects afterwards. The first is done using the generator query found in Table 4.3, the latter using the one in Table 4.2. By construction, the generator query only returns entities, which have the required information in the question stored in the KB. This is achieved by not only searching for names or insurance IDs using '`?s gleif-L1:hasLegalName ?a`' or '`?x eiopa-Base:hasInsuranceUndertakingID ?a`', but also adding the condition '`?e eiopa-Base:hasEUCountryWhereEntityOperates []`' to the generator query. The condition states the returned entity must have some data for the predicate `eiopa-Base:hasEUCountryWhereEntityOperates`. The left and right square bracket (‘`[]`’) act as wildcard, so there only has to be some data as object to the predicate, but we do not require further specification what it should be. This wildcard is used at the end of the generator query in Table 4.2 and 4.3 to ensure that the requested information is present, but no concrete value needs to be satisfied. An example for a filled template, now called a language pair can be found in Table 4.4 and Table 4.5.

Question	In what country does w1981 operate?
Query	select distinct ?o where {?e eiopa-Base:hasIdentifyingName "w1981". ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o.}

Table 4.4: Language Pair using ID

Question	In what country does aegon levensverzekering n.v. operate?
Query	select distinct ?o where {?e eiopa-Base:hasIdentifyingName "aegon levensverzekering n.v.". ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o.}

Table 4.5: Language Pair using Name

Some adjustments to the procedure of [24] had to be made, since the underlying KBs are of different structure. The DBpedia KB uses human-readable URI as subjects. For example, the monument of Carew Cross in Wales can be found at `https://dbpedia.org/resource/Carew_Cross`. These human-readable URI simplify the queries, since no condition on the name as an object must be added. Furthermore, since the work described in Chapter 3 does not explicitly focus on the presence of rare or unknown names, the model just learns the mapping between the name and URI. However, entities might only be present a restricted number of times and hence it is hard to correctly learn the mappings. This might result in mistakes in entity matching, which was noticed in [32], [24] and [19]. The URI of subjects are not human–readable in the KB, we made for the EIOPA data and it might be hard for an NMT model to create the mapping between a name and the URI, which might look like this `https://rdf.eiopa.europe.eu/L1-data/IURI-De-Nederlandsche-Bank-H0074`. As our work focuses also on the use of unknown names, we changed the structure of the queries. The queries have an extra condition for the name and accordingly only use the URI of the subject implicitly. As consequence, the NMT model only has to map the name of an entity to the correct position in the query. Additionally, the model has no possibility to separate between the use of an insurance ID or an entity name, which it might not be able to, especially if all entity objects are treated as an OOV word in some models.

Most names of insurance companies use small and capital letters, often in a non-logical manner, which might be difficult for the model to correct translate. Therefore, when the entity names and ID codes are added to the question or query, we uncapitalize the object. This step slightly simplifies the task of the model, since the translation of names is not case-sensitive. Additionally, square brackets, round brackets and quotation marks are removed, since we want to prevent inference of special symbols in names with the SPARQL structure. SPARQL queries contain an intrinsic structure with different brackets and quotation marks. The presence of other special symbols also present in SPARQL queries might lead to struggles in the encoding or decoding process described in Section 4.2. Nevertheless, we need to add double quotation marks around the name for the correct working of the SPARQL query. This is only done for entity objects in the query, but not in the question. This improves robustness, since the model has to actively recognize what is the entity object in the question.

In first instance, we constructed the queries to use the full legal name, but this is error prone. Already if there is a small element missing in the name, the SPARQL query will not return any correct result. As users might use short-forms of names and copy networks are a possible model architecture (see 2.3.3), some adjustments had to be made to the KB. We added a new predicate called '`eiopa-Base:hasIdentifyingName`'. The objects to this predicate are short forms of the entity names used in daily work at DNB and the insurance ID. It can also be observed in Table 4.3 and 4.2 that for generating the language pair the predicate '`gleif-L1:hasLegalName`', respective '`eiopa-Base:hasInsuranceUndertakingID`' are used to fill the placeholder, but in the search query in both cases '`eiopa-Base:hasIdentifyingName`' is used to find the required information. Furthermore, we restrict our used names to the Netherlands and Europe. The purpose of the latter is on the on hand to only work with a normal alphabet and prevent possible issues with cyrillic or greek alphabet. This might be occurring, if we sample names from whole Europe, which are contained in the KB. On the other hand, insurance supervisors are mainly concerned with dutch insurance companies, so the model should work optimally with these types of names.

When generating the data, all questions are directly tokenized using Moses tokenizer with english setting. The tokens are then merged together seperated by spaces. This process cannot be used for the SPARQL queries, since certain elements which should stay together, e.g. variables like '`?a`'. Therefore, a special tokenization and encoding process is used described in Section 4.2. When generating train or validation data, we also generate so called 'raw' datasets, where the placeholder is replaced by an space. These files can be used to create dictionaries with no entity names or insurance IDs contained. The originally present option to use distinct classes for a variable and the option to prioritize the use of certain entities is not used anymore, since our KB is quite small and does not contain the required data for usage prioritization. For details on this, we refer to [11].

4.2 Tokenization and SPARQL encoding

Tokenization or segmentation is an import part in MT. “It covers simple processes such as separating punctuation from words (tokenization), splitting words in subparts based on their frequency or more sophisticated processes such as applying morphological knowledge” [5]. The splitting of subwords is handled separately and we focus in this section on the separation of words and punctuation. This process is commonly named tokenization and the result of tokenizing a sequence is denoted as tokens. Domingo et al. concluded that tokenization has great benefits for NMT translations [5]. In programming terms, this usually means that a string sequence, i.e. a sentence, is split into a list of strings. Thus, each word or punctuation symbol is one element in this list. It is then possible to merge back the tokens, all seperated by a space, as one string.

There are many standard tokenizer libraries available and these can easily applied to the input side and the questions in natural language. We use the python implementation of Moses, called Sacremoses [3]. Moses is an open-source toolkit for statistical MT and with it also comes an tokenizer, which is widely used. The tokenizer separates words from punctuation, but it preserves special structures liked URLs or dates. Additionally, special symbols like quotes are transformed to unicode [14, 5]. For the pre-trained XLMR model, the sentencepiece [4] tokenizer was required and therefore used (see Section 4.3.6).

The matter is different for the translation side with SPARQL queries. SPARQL queries have an internal structure and combine query language with specific elements of the RDF data. For many elements of the queries, punctuation and letters are joined together and should rather not be split. For example, variables in SPARQL are denoted by a word or letter and a question mark, ‘`?var`’. Also predicates are merged together by a colon and dashes, ‘`eiopa-Base:hasIdentifyingName`’. We suppose that it is easier for a NMT model to produce correct translations when important elements of a SPARQL query stay together and are not split. Therefore, we take over the approach of Yin et al. and Soru et al. to encode SPARQL queries into a sequence by performing replacements [24, 32]. Due to the difference in the KB, some elements of the encoding process were omitted, replaced or newly added.

Due to the use of the encoding process, the NMT models do not directly translate to SPARQL, but rather produce translations in the encoded SPARQL format, which then are decoded. All special symbols in the queries are replaced by a verbal description, i.e. ’{‘ turns into ‘`bracket_open`’. The full list of all replacements can be seen in Table 4.6. The encoding process is done using string replacements and turns the queries into a sequence with only standard letters and underscores. This process can be easily reversed by doing reverse replacements, i.e. doing the string replacements in the opposite direction.

[3]`https://github.com/alvations/sacremoses`
[4]`https://github.com/google/sentencepiece`

Original	Encoding
`'eiopa-Base:'`	`'eiopa_base_'`
`'gleif-L1:'`	`'gleif_l1_'`
`'gleif-Base:'`	`'gleif_base_'`
`'n.v.'`	`'n_v'`
`'u.a.'`	`'u_a'`
`'b.v.'`	`'b_v'`
`' ( '`	`'  par_open  '`
`' ) '`	`'  par_close  '`
`'('`	`' attr_open '`
`') '`	`' attr_close '`
`'{'`	`' brack_open '`
`'}'`	`' brack_close '`
`' . '`	`' sep_dot '`
`'. '`	`' sep_dot '`
`'?'`	`'var_'`
`'*'`	`'wildcard'`
`' <= '`	`' math_leq '`
`' >= '`	`' math_geq '`
`' < '`	`' math_lt '`
`' > '`	`' math_gt '`
`' "'`	`' quot_mark_l '`
`'" '`	`' quot_mark_r '`
`'"'`	`' quot_mark_n '`

Table 4.6: String Replacements for SPARQL Encoding

As said before, some adaptions to the encoding process were made. As described in Section 4.1, the entity objects are enclosed by quotation marks. This also need to be encoded and we use two different key words for this. One for the left quotation mark (‘`quot_mark_l`’) and one for the right (‘`quot_mark_r`’). This enables us to correctly use spaces, when decoding the sequence. An extra space between the entity object and the quotation mark would lead to no results when executing the query. Moreover, we also add the abbreviations of dutch legal entity names like ‘b.v.’ for ‘besloten venootschap’. If these abbreviations are not encoded using a special keyword like ‘`b_v`’, we get a different encoding using ‘`sep_dot`’, which results in wrong decoded sequences. As last, the namespaces of our KB replace the namespaces used in the KB of DBpedia.

4.3 Models

We propose different model architectures to handle the task of translating questions in natural language to the query language SPARQL. The focus of our approach is on the correct translation of the entity objects, which are a crucial element of generating queries, which will give the correct result.

4.3.1 Transformer Model

At first a standard NMT model using a Transformer architecture is proposed. This mimics the approach taken in the Transformer model of [32]. A more detailed explanation on Transformer models can be found in Section 2.2.4. We decide to use transformer_iwslt_de_en architecture provided in the fairseq package [5] and explain in the following the high-level design of the model compared to the original transformer architecture. This model consists out of of 6 encoder and decoder stacked layers. However, it is slightly smaller by only using 4 attention head for multi-head attention, compared to 6 in the original architecture by [28]. The reason is that our dataset is rather small and contains comparably shorter sequences. The embeddings have the same size of 512, but the elements of the feed-forward network are halved in size to 1024. Accordingly the model is a bit lighter, needs less memory and computational resources and is hence quicker to train on low-resource environment. The data is trained with the generated data (compare Section 4.1) with no further special pre-processing applied and fed to the model on a word-level basis.

4.3.2 Replacing the Out-of-Vocabulary Words

As explained in Section 2.3.1, there exist several approaches to improve translation of rare and unknown words. As first, we use a model using an alignment mechanism to replace OOV words in the output sequence. We integrate the approach into the transformer_iwslt_de_en architecture described above. As a Transformer model operates exclusively and cross- and self-attention and attention relying on alignments, no deep changes to the architecture have to be made. The method is to average over all attention matrices of all layers and heads and turn the resulting continuous probability distribution into discrete alignments. When generating outputs, we replace the OOV token `<unk>` by the aligned word in the source sequence. Unknown words are mainly encountered, when we test the model on different names than present in the train set, but this depends on the experimental setup (see Chapter 5). The functionality to replace OOV words in the target sequence and output the alignments between source and target words is provided in the fairseq framework. However, some bugs in the code needed to be fixed to make it work and we implement a small change in the source code of fairseq by enabling the '–replace-unk' option to also access words in the raw source sequence. The reason is that some entity names might be unknown at test time and hence no present in neither the source nor the target dictionary. In the original way of fairseq replacing the unknown word,

[5] `https://github.com/pytorch/fairseq/blob/0972dde844e39540faf53b6d9afe76b38c7e2fd6/fairseq/models/transformer.py`

we might get an OOV token replaced by an OOV token, if the aligned word is not present in the source dictionary.

4.3.3 Transformer and Aligning

Using the integrated alignments of a Transformer architecture seems promising, but Garg et al. claim that the alignments of a standard Transformer model are more concerned with the context of a target word than with the exact aligned source word [7]. Therefore, the generated translations might not completely resemble what is usually seen as word alignments between source and target sequences. Garg et al. propose a new model which is focusing on correct translations, but also on producing correct word alignments. They implement a multi-task loss function and train one attention head to learn alignments. The results show the model to be competitive with state-of-the-art SMT alignment methods, while not sacrificing correctness of the translation [7].

The approach of [7] seems promising and might improve the performance for correctly translating the entity objects. We incorporate the model into our training pipeline and train it on the generated data on word-level basis as only then unknown words will likely be present. The architectural design of the Transformer is similar to above.

4.3.4 Subword Model

A different approach for improving translation of entity objects is to let the Transformer model operate on a subword-level. We use subword algorithm provided by Sennrich et al. and the provided python implementation[6]. The implementation is as explained in Section 2.3.1, with the small change in the end-of-word token to ensure non-ambiguity in what is a subword and a final word. We apply the default number of merge operations being 1000. The BPE are learned on the merged dictionary of the input and output corpus to prevent certain encodings only being present in one language. The learned encodings are applied to the source and target corpus and using the train data, we create new dictionaries of the used subwords. This is then passed on to the model. When generating the sequences, the subwords are concatenated again to create the output.

4.3.5 Pointer-Generator Model

In Section 2.3.3 we stated the work on copy/pointer networks to deal with rare and OOV words. The pointer-generator approach of [22] is the most promising and advanced approach and hence we chose to implement this and give more

[6]`https://github.com/rsennrich/subword-nmt`

explanation of the workings in the following. Enarvi et al. provided the adaption to Transformer models for seq2seq task.

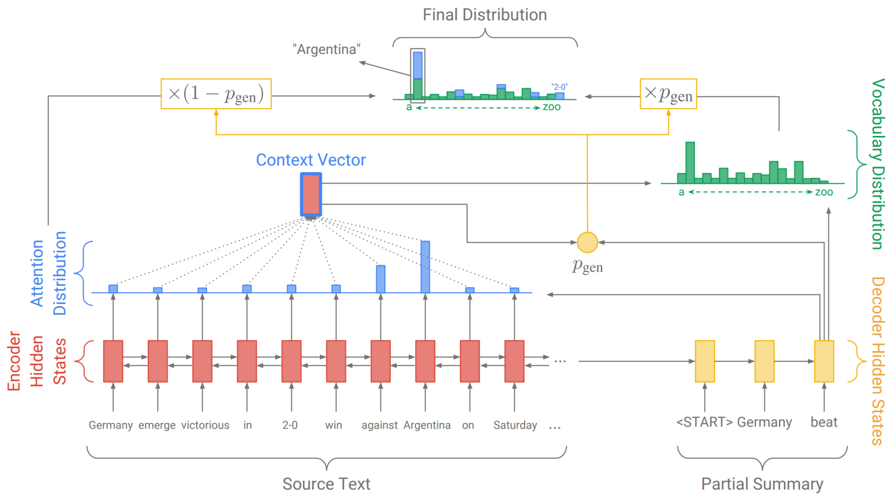

Figure 4.1: Architecture of Pointer-Generator Model [22]

The idea of the pointer-generator model is the possibility to decide whether to copy a word from the source sequence or generate it from the dictionary. The complete architecture is shown in Figure 4.1 The authors implement a single-layer bidirectional LSTM as encoder (red in 4.1) and feed the tokens w_i of the input one-by-one into it [22]. This encoder outputs a sequence of hidden states h_i. On each step t of the output sequence, the decoder (a single-layer unidirectional LSTM) uses the information about the previous word to calculate the decoder hidden state s_t. The encoder and decoder hidden state(s) are used to calculate the *attention distribution* as proposed by [1]:

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + b_{attn}) \tag{4.1}$$

$$a_t = \text{softmax}(e^t) \tag{4.2}$$

v, W_h, W_s and b_{attn} are learnable parameters. This attention distribution (shown in blue in Figure 4.1) indicates the model where the relevant parts for the next generated word are to be found. Accordingly, using the attention distribution and Equation 4.3 a weighted sum of the encoder hidden states is calculated called the *context vector* h_t^*.

$$h_t^* = \sum_i a_i^t h_i \tag{4.3}$$

The context vector summarizes all the information about what has been read at the step t from the source sequence to a fixed-size vector. To decide whether to point a word from the source or generate a word from the dictionary, we define the *generation probability* $p_{gen} \in [0, 1]$ in Equation 4.4.

$$p_{gen} = \sigma(w_{h*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr}) \tag{4.4}$$

The calculation of p_{gen} for step t is learned by adjusting the learnable parameters w_{h*}, w_s, w_x and b_{ptr}. σ represents the sigmoid function, i.e. $\sigma(x) = \frac{1}{1+\exp^{-x})}$ and x_t the decoder input. Next, let $P(w)$ define the extended vocabulary w as union of the target dictionary and all words in the source sequence(s).

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t \tag{4.5}$$

$P(w)$ is the final distribution of the extended vocabulary (green/blue on top in Figure 4.1). Note, that if w is OOV, then $P_{vocab}(w) = 0$. On the opposite, if w is not present in the input sequence the sum over the attention distribution will be zero. By, using this approach, the pointer-generator model is able to produce OOV words to the output sequence [22].

See et al. additionally use coverage to prevent repetition, which is a common problem in seq2seq problems. During our research, repetition also occurred and parts of entity object was added multiple times to the candidate translation. See et al. include coverage by maintaining a *coverage vector* c^t, which is the sum over the attention distributions of the previous timesteps:

$$c^t = \sum_{t'=0}^{t-1} a^{t'} \tag{4.6}$$

c^t represents the amount of coverage the words in the source sequence have received already. The coverage vector is introduced as extra input to the attention mechanism (Equation 4.7) and hence the current decision of the attention distribution (where to look next for the relevant word in the source sequence) is influenced by the previous decisions, summarized in c^t.

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + W_c c_i^t + b_{attn}) \tag{4.7}$$

Equation 4.7 is a modified version of Equation 4.1. Coverage is also included by using a special coverage loss, which penalizes the model for attending to positions with already high coverage [22].

Since the introduction of Transformer models [28], these became the standard method for seq2seq tasks. Therefore, the pointer-generator approach is integrated into a Transformer architecture by [6]. The authors switch from sinusoidal position information to relative position representation, since they found this to have better performance [6]. In the Transformer model, in each sublayer of the decoder several encoder-decoder attention distributions are generated. This happens in every head of the multi-head attention and one single head or the average of all heads can be used for the pointing mechanism. We take over the approach of Enarvi et al. to dedicate one single head for the pointing task of one attention head. This attention head will then also learn to correctly learn the alignments similar to the techniques described by [7]. The results of [7] are used to draw the conclusion that the penultimate attention layer naturally learns alignment-like structures. The rest of the structure is the same as in the normal transformer_iwslt_de_en architecture.

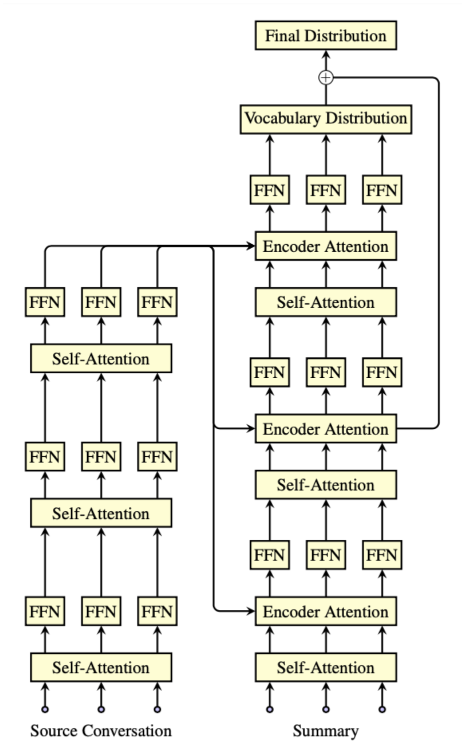

Figure 4.2: Pointer-Generator Transformer Architecture [6]

The pointer-generator mechanism for Transformers can also be observed from Figure 4.2. The outputs of the penultimate layer are interpolated with the vocabulary distribution by the predicted generation probability P_{gen}. This then either outputs a word from the vocabulary or points a word from the source sequence. The calculations are the same as when a RNN is used. In Figure 4.2, the residual connections and layer normalization layers are omitted for clarity [6].

The design of fairseq demands a additional step for pre- and post-processing. In fairseq it is not possible for the model to copy tokens from the original raw input sequence, but it can only access an encoded input sequence. This encoded input sequence is already processed with the source dictionary and hence does not contain OOV words anymore. In order to solve this issue and let the model be able to produce OOV words to the input sequence, Enarvi et al. propose to add a number of tokens to the source dictionary of style `<unk-N>` [7]. Before the input is passed to the model, all OOV tokens are replaced by such a `<unk-N>` token, where the N represents the position in the raw input sequence. Hence, when the raw input sequence is prepared by the framework as input to the model, the OOV tokens are now seen as part of the source dictionary. Nevertheless, when the architecture itself processes the sequence, all `<unk-N>` are mapped to the standard `<unk>` for OOV words. Hence, the model is not just learning to copy when a `<unk-N>` is present, but decides based on the context vector and decoder hidden state like explained above. The `<unk-N>` is only an auxiliary token, which is used to prevent the copy mechanism not able to copy an OOV word as in fairseq the decoder cannot access the raw input sequence. If the pointer-generator model decides to point and produce a `<unk-N>` in the output sequence, we use the position stored in N to replace `<unk-N>` by the token in the raw input sequence. The implementation of this approach was available in fairseq and after small adaptations and adding some functions to match with our concrete pipeline, we can reuse the model and processing scripts.

4.3.6 XLMR Model

As one objective of the research is to build a translation model to SPARQL from english and dutch, we propose a encoder-decoder Transformer architecture using the the XLM-R model as encoder. The idea is to make use of the potential of pre-trained language understanding the in from of token representations. We include XLM-R model as encoder and add a Transformer decoding stack consisting of 6 layers. Hence, the decoder has a similar set-up as the other Transformer model used. We argue that the layers should be enough to correctly decode the learned representations to encoded SPARQL sequences. Additionally, as the XLM-R models is large and the computational resources are constrained, it is not reasonable to use a decoded of similar size as the XLM-R encoder. This is should also not be necessary as the task for the model is much more simple, only 1 language compared to 2, or actually 100 in the pre-training process. The XLM-R model operates on subwords generated by sentencepiece, which is quite similar to subword-nmt. We use the same sentencepiece model for input and output, which is not optimal, since it is the word encodings are accordingly not trained on SPARQL. However, the setup of fairseq require this and for a first test it should be acceptable.

[7]`https://github.com/pytorch/fairseq/blob/master/examples/pointer_generator/README.md`

4.4 Evaluation Metrics

We use several evaluation metrics to evaluate the quality of the translations with a focus on the translation itself and also a result-oriented focus. Standard techniques for NMT like the BLEU score and string matching accuracy are used. Due to the translations being queries, we also test whether the translation and reference query return similar answers. As our research is especially focused on rare and unknown names, we also evaluate whether the entity object in the reference and the translation do match. All metrics are explained in detail in the following.

4.4.1 BLEU score

For MT tasks it is standard to evaluate the quality of translations using the BLEU score (for details see Section 2.4). The ability to chose certain elements of the BLEU score as hyperparameter, i.e. w_n and N_{max}, creates a problem for comparability of BLEU scores [21]. Post therefore called for clarity in reporting of BLEU scores. He claims that there does not in fact exist 'the' BLEU score and the chosen parameters are vastly not reported. Also, preprocessing schemes have a large effect on BLEU scores. He proposes a common approach for calculating and reporting BLEU scores and supplies a new python package, called SacreBLEU[8]. The method of SacreBLEU is based on the BLEU scheme used by the annual Conference on Machine Translation (WMT) and has parameters fixed. Additionally, it is specified that detokenized should be used for evaluation. In order to ensure easy reproducibility and comparability of our results, we use the SacreBLEU python script for calculating BLEU scores. Papineni et al. stated that evaluation on only one reference is a valid method, so this does not pose a problem for our research.

4.4.2 String Similarity

BLEU is the most-used metric for NMT problems to rate the quality of translations and it has been shown to have high correlation with human evaluation [20]. In spite of that, Tran et al. conclude that BLEU scores do not reflect well translation quality for problems concerning code as there is weak correlation with the semantic correctness of translated code [27]. They proved this for different statistical MT task concerned with code migration, where code is translated from one programming language to a different one. The reason is that BLEU scores do not fully capture the correctness of syntax and dependencies present in code. The result of Tran et al. conclusion is relevant for our research as we translate to SPARQL, which is similar to a programming language and can be seen as 'code'.

[8]https://github.com/mjpost/sacrebleu

The authors of [27] claim that a metric, which measures the results in a higher abstraction level, is the better metric for reflecting semantic accuracy. They propose the metric RUBY, which measures the quality of translation in a multi-layer manner at a lexical, syntactical and semantic level. As a SPARQL query is not really code, we must restrict ourselves to the lexical method. Nevertheless, this metric should have a slightly higher correlation with semantic correctness than the BLEU score, according to the results of [27].

We implement String Similarity (STS) as metric to compare the similarity of reference and candidate translation represented by a sequence of code tokens. The similarity between reference R and translation T is calculated using Equation 4.8

$$\mathrm{STS}(R,T) = 1 - \frac{\mathrm{SED}(R,T)}{\max(|R|,|T|)} \tag{4.8}$$

String Edit Distance (SED)(R,T) is the string edit distance between reference R and translation T. Tran et al. do not exactly specify their calculation method for SED and therefore we decide to use Levenshtein distance [15]. The Levenshtein distance measures how many elements of a string need to be substituted, deleted or added to transform the one string into another. We use the `editdistance` python package, which is an efficient C++ implementation of an adapted version of the Levenshtein distance by Hyrö [13]. This implementation is also able to work on token basis instead of character level. Accordingly, we calculate SED as the number of tokens to be substituted, deleted or added to transform the translation T into the reference R. $|R|$ and $|T|$ refer to the number of tokens in the respective sequence. STS is calculated for every translation and we report the mean of all similarity scores over the whole test set. A closer matching translation will score close to a value of 1.

4.4.3 Name Match Accuracy

Our research has a focus on the problem of translating entity objects in a correct way. The entity names or id codes are rare or even unknown at test time and this poses a special challenged for a NMT model. To evaluate our progress with respect to unknown names, we design a special metric. As explained in Section 4.1, the entity object is surrounded by quotation marks in the query. Our metric is constructed to extract the object surrounded by quotation marks from decoded SPARQL query (compare Section 4.2) and evaluate on it/.

First, the entity object from the reference is extracted and used to construct a query, which returns all objects stored with the predicate '`eiopa-Base:hasIdentifyingName`'. As explained before, most insurance companies have a long legal name and in daily use at DNB shorter forms do exist and are widely used. These name variations and the insurance id code are stored under '`hasIdentifyingName`'. As last step, the entity object from the generated translation is extracted and we check whether this name is amongst the entity objects returned by the query described before. We intentionally do not compare the objects of translation and reference directly, since it might happen that the model turns a short form into the full legal name or an insurance ID into a name. By first using a query to get all identifying names, we prevent false negative results, if the object describes the correct entity, but just in a different way. The concrete metric is calculated by dividing all correct matches by the total number queries examined.

4.4.4 Query Result Accuracy

As the translations of our model are SPARQL queries, it is also of interest whether the translations return the expected result, when executed on the KB. It is also possible that two different queries return the same result, for example both the GLEIF and EIOPA dataset contain the country of an insurance. Hence, there exists different queries to get the country of a respective insurance. In this case, the direct evaluation of the translation reports a bad translation, which is from a result-oriented perspective not completely true. We therefore include query result accuracy as metric.

At test time, we execute the translation and reference query on the KB and compare the returned results. As some queries should return several items as result, we test for each item of the reference result whether it is contained in the result items of the translation. If an item is contained in the translation result, we count it as correct and if not as false. Hence, it is not counted as false, if the translation query returns too many result items, since an user will not mind to get more information than requested. In case, the execution of the translated query results in an error or an empty result list, this is as counted as false. The query result accuracy is then calculated by dividing the number of correct results by the number of correct and false results added together. Also the number of correct and false cases is reported. Due to the design of the metric, it might happen that the sum of false and correct cases is larger than the number of queries. The reason is that we test each result item of the reference translation separately and there might be multiple result items for one query.

A limitation of the query result accuracy is a dependency on the correct name match (Section 4.4.3). If the name is wrongly matched between translation and reference, the query probably also does not return the correct result. Hence, the accuracy of the present metric is capped by the name match accuracy. However, by the difference between the two metrics we can observe, what portion of queries had a correct name match, but some deficiency led to a poor result.

Chapter 5

Experimental Setups

In the following, we explain the concrete set-ups used to train and evaluate the model architectures described in Section 4.3. Additionally, more information about the datasets and the framework is given.

5.1 Framework

The pipeline for evaluating the models and generating the language data is written in python. For the NMT process, we use the fairseq framework. This is a sequence modelling toolkit by Facebook AI Research [18], which is well optimized and easily extensible. Furthermore, many state-of-the-art work is present on the repo and also pre-trained models are available. The toolkit is based on PyTorch and models can be easily trained and evaluated using commands from the command line. As described in Chapter 4, the toolkit is not without errors and accordingly some mistakes had to be corrected to make certain models work. The concrete version of fairseq used including adaptions and corrections is available on github [1]. Models are trained on a high-perfomance computing center using standard CPUs.

5.2 Datasets

We generate our datasets using templates as described in Section 4.1. An overview of all template sets can be found on the github repo [2]. For every template, we generate 130 examples, so we use each template 130 times with

[1] `https://github.com/jm-glowienke/fairseq/tree/ca45353322f92776e34a7308bf3fab75af9c1d50`

[2] `https://github.com/DeNederlandscheBank/nqm/tree/main/data/eiopa/1_external`

different names. For the train and validation test set we use combined 69 templates and split the resulting dataset. 80% are used for training the dataset and 20% for validation during training. The template set contains for each query up to 5 different questions, how we expect an user asking for this information. This should result in a more robust performance on unknown questions. Additionally, the queries are always generated using twice with one time the insurance id code and one time the name of the insurance, except for queries where this is not logical to do, e.g. question about the insurance id code will not be asked using the insurance id. We generate different train and validation sets, where the names and insurance id are part of the dictionary or not. For generating a dataset with known and unknown names, we use two sets of templates and only produce 65 examples per template. The difference is that the generator queries of one template set only search for dutch insurances, while the other one is returning german insurances. The resulting language pairs from the two templates sets are merged, but only the dutch insurance names are added to the vocabulary, while the german ones are treated as OOV. This serves the purpose of testing, which training data leads to better performance. When generating a train and validation set with only unknown or known insurance names, we only use dutch insurances. Either way the train set consists of 7176 language pairs and the validation set of 1794.

For testing the performance of a model, we use 4 different test sets to calculate the evaluation metrics. For each test set, we have a set of templates to also generate this data with 130 examples per template. For test set 1, we use 9 templates, where each template consists out of a query also present in the training data, but a new question not seen before. The 1170 language pairs are generated using dutch insurance names. Test set 2 exhibits the biggest challenge for the models to handle, as we use a combination of two known queries and questions. Thus we observe whether the models are able to produce queries returning two objects at once and exhibit some generative power for new queries. Only one template is used for this. For test set 3 and 4, we both use questions and queries seen during training. Test set 3 is generated using german insurance names and test set 4 using dutch insurances. Using these two test sets it is possible to solely compare the performance on the different set of names. We use 9 templates to generate the test datasets 3 and 4.

We additionally perform an experiment with bilingual language pairs. We test whether the model can handle this well and enable us to ask questions in english and dutch. The dutch template sets are in setup and in terms of queries identical to the english template sets and have numbers 5–8. Only the train set is slightly smaller as certain questions are not translatable to proper dutch.

5.3 Model Setups

For each of the model architectures/types described inSection 4.3 we test different setups in terms of input to see what works best. This is also to answer our research question whether the handling of rare and unknown entity is best when we this names are part of the dictionary for training or not. An overview of all models can be seen in Table 5.1.

Model Name	Type	Data
A (ALPHA)	Transformer standard	All names known
B (BRAVO)	Transformer standard	Unknown and known names
C (CHARLIE)	Transformer standard with alignment replacement	All names known
D (DELTA)	Transformer standard with alignment replacement	Unknown and known names
C2 (CHARLIE2)	Alignment Transformer model with alignment replacement	All names known
D2 (DELTA2)	Alignment Transformer model with alignment replacement	Unknown and known names
E (ECHO)	Subwords Transformer model	All names unknown
F (FOXTROTT)	Subwords Transformer model	Unknown and known names
G (GOLF)	Pointer-generator model	Unknown and known names
H (HOTEL)	Pointer-generator model	All names unknown
I (INDIA)	Pointer-generator and subwords model	All names unknown
J (JULIETT)	Pointer-generator and subwords model	Unknown and known names
K1 (KILO1)	Subwords Transformer model	Unknown and known names, bilingual
K2 (KILO2)	Pointer-generator model	Unknown and known names, bilingual
L (LIMA)	XLMR-iwslt model	Unknown names

Table 5.1: Overview of all tested models

Basically, for all model architectures and methods for dealing unknown words described before, we train them on two datasets. Either with all names treated as known, unknown or half known and half unknown. Known or unknown means whether the entity objects are treated as part of the model vocabulary or not. The models are then trained using a standard hyper-parameter setting provided by fairseq[3]. When training the models we use a maximum of 200 epochs with early stopping. If there is for 20 epochs no improvement for the BLEU score on the validation set, the training is stopped. Accordingly, the models are trained in a range of 60–90 epochs.

[3]Detailed shell scripts for each model setup can be found on the github: https://github.com/DeNederlandscheBank/nqm

Chapter 6

Results & Discussion

We evaluate the performance of our model experiments using the evaluation metrics described in Section 4.4 and calculate these metrics for the 4 different test sets used. Table 6.1 gives an overview of all metrics, which we will discuss in the following in more detail. Next to analysing the metrics, we also do a qualitative analysis of some exemplary translations. As the translation task is not so difficult for the large Transformer models, the Name Match Accuracy and Query Result Accuracy are of most interest, although the values of these two metrics are highly correlated. The latter metric is of special interest from an user perspective, which is interested the most that a correct result is returned by the pipeline.

A general conclusion from the values in Table 6.1 is that models all score consistently lower on test set 2, which requires the generation of a new query, which is a combination of two queries from the train set. All models are not able to generate such a query correctly and mostly build a query, which gives half of the answer on the question asked. This shows that the chosen models are not able to produce new queries by themselves and only perform well in mapping some question to a query seen during training. Another general problem is punctuation in names. This was already touched on in Section 4.1 and we tried to take control of the problem by removing most punctuation and special symbols from names. Nevertheless, dots are necessary in abbreviations, which are frequently used for legal forms in company names. As dots are also essential for the separation of the triplets in SPARQL queries, this leads to errors. The dots are encoded to the token 'sep_dot', if they are at the end of a word, but in the reverse replacement an extra space is inserted, which is relevant for the separation of the SPARQL queries. However, abbreviations also get an extra space so 'e.o.c.' becomes 'e.o.c .' which leads to problems. We covered some legal form abbreviations and used a special encoding token for these, but of course it is impossible to cover all of these and this would also not be beneficial as the model should rather handle it automatically.

	BLEU				Name Match Accuracy (%)				Mean String Similarity (%)				Query Result Accuracy (%)			
Model	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
ALPHA	94,06	38,50	92,15	99,78	96,32	90,77	0,00	96,24	99,58	95,11	98,79	99,96	98,45	46,09	10,94	98,63
BRAVO	91,23	37,45	85,05	100,00	77,09	0,00	0,00	99,49	99,36	94,27	98,61	100,00	77,62	0,00	0,00	100,00
CHARLIE	94,06	38,50	92,15	99,78	96,32	90,77	0,00	96,24	99,58	95,11	98,79	99,96	98,45	46,09	10,94	98,63
CHARLIE2	94,06	38,50	92,15	99,78	96,32	90,77	0,00	96,24	99,58	95,11	98,79	99,96	98,45	46,09	10,94	98,63
DELTA	92,84	34,86	93,45	100,00	77,09	0,00	22,22	99,49	99,37	94,30	98,83	100,00	77,62	0,00	22,93	100,00
DELTA2	92,84	34,86	93,45	100,00	77,09	0,00	22,22	99,49	99,37	94,30	98,83	100,00	77,62	0,00	22,93	100,00
ECHO	91,36	70,34	90,07	98,83	75,88	31,54	0,00	83,58	99,24	96,71	98,51	99,81	81,12	0,41	8,53	87,71
FOXTROTT	94,11	66,94	98,42	99,68	97,26	21,54	75,00	95,89	99,59	96,46	99,78	99,95	97,59	0,91	83,45	97,08
GOLF	93,20	52,03	99,42	99,65	73,29	0,00	91,62	99,15	99,52	95,40	99,91	99,98	73,54	0,00	94,70	97,07
HOTEL	85,22	66,45	95,81	93,87	24,02	0,00	38,65	36,35	98,86	96,19	99,65	99,62	24,14	0,00	35,45	32,47
INDIA	90,36	39,06	86,16	99,74	65,87	0,00	7,26	95,98	99,23	93,86	98,14	99,97	62,83	0,00	7,19	95,70
JULIETT	93,07	39,48	98,41	99,69	72,70	0,00	84,19	99,66	99,37	94,10	99,84	99,98	53,78	0,00	80,84	97,08

Table 6.1: Overview of Results for Experiments

	BLEU				Name Match Accuracy (%)				Mean String Similarity (%)				Query Result Accuracy (%)			
Model	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
KILO1	93,10	36,65	95,77	96,92	90,43	52,31	77,78	98,46	99,45	94,71	99,25	99,46	90,71	26,98	75,52	88,64
KILO2	93,44	36,66	98,64	99,12	79,32	0,00	87,05	99,31	99,49	94,32	99,85	99,93	75,09	0,00	89,36	91,74
	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8
KILO1	93,02	38,63	95,76	96,92	87,18	95,38	76,58	99,15	99,14	95,13	99,25	99,46	75,95	48,44	74,65	88,16
KILO2	95,14	38,14	96,96	99,19	85,81	64,62	88,72	99,32	99,64	95,02	99,50	99,94	73,93	39,25	68,85	92,96

Table 6.2: Results of Bilingual Models

6.1 Model A and B

Model A and B consist out of a standard Transformer model with no special preprocessing steps. Model A archives BLEU scores of above 90 for test set 3 and 4, which consist out of known questions. On test set 1, which has unknown questions for known queries, the performance is slightly lower with roughly 5 points less than for test set 4. The performance on an unknown query (test set 2) is considerably lower, which tells us that this model is not able to combine two known questions to the correct query by itself. This conclusion will hold for most models. As Model A was only trained on dutch insurance entities, it does not perform well on german insurance names (test 3). As the names are not part of the dictionary the model treats them as OOV and when translating does not known what name there is on this position. It archives zero correct name matches and inherently only 10,94% correct query results. The last result is actually quite interesting, since one might expect that if there is no correct name match, then there should be no correct result. A detailed analysis of the translations and query results gives a possible explanation. We see that almost all insurance objects are translated to a random entity object , which is apparently the solution for the network to resolve the issue of having an OOV token at the position, where it expects the name or insurance ID of the entity. However, the questions posed about the cross-border status of an insurance company might yield the same result "Domestic undertaking" for the german insurance and the random dutch insurance from the vocabulary. This example is shown in Table 6.3. Hence, although the query is searching for information about a complete wrong insurance company, there still might be a match between the results of reference and translation. This appears to the evaluation metric as correct, although it is not really correct. Therefore, it is important to pay attention what entity object the translation is containing, when using our model for a translate and query tool. This observation indicates the model is able to produce semantically and syntactically correct queries, even if the entity object appears unknown to it.

Reference	SELECT DISTINCT ?o WHERE {?e eiopa-Base:hasIdentifyingName "debeka lebensversicherungsverein auf gegenseitigkeit sitz koblenz am rhein" . ?e eiopa-Base:hasCrossBorderStatus ?o . }
Reference Result	Domestic Undertaking
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "w1875" . ?e eiopa-Base:hasCrossBorderStatus ?o . }
Translation Result	Domestic Undertaking

Table 6.3: Example for Correct Result despite Entity Mismatch

Model B had the same set-up as A, but its train set also contained german entity objects. Nevertheless, these are not part of the dictionary and the performance is even slightly worse. The performance on test set 3 is worse as the model places the `<unk>` in the translations and hence does not profit from the accidental results match described before. In hindsight, training this model set-up on known and unknown names is not really beneficial.

6.2 Model C, C2, D and D2

Model C, C2, D and D2 all try to handle unknown words by replacing OOV tokens in the translation by the aligned word of the source sequence. The evaluation metrics have the same values for Model C as for Model A. The analysis of the translations show that the model is not learning to place the OOV token in the translation but rather places a random entity object from the dictionary similar to before. When training on known and unknown names, we can observe an improvement in performance. Model D scores an average BLEU score on all 4 test sets of 80,29 compared to 78,43 for Model B. Additionally the name match accuracy is raised to 22,22% compared to 0%. However, this accuracy is too low to use the model for translating unknown names. The examples in Table 6.4 indicate that a problem with coverage is present. The model is not able to well generalize and translate the varying number of OOV tokens correctly. Apparently, in the train set mostly three OOV tokens were present as most insurance names consist out of 3 parts. Accordingly, at test time a present `<unk>` might be translated with several tokens. Additionally, the alignments between target and source words are off. In the second example, the model is incorrectly aligning to the word "the" in the source sequence. An analysis of the discrete alignment pairs show that the model does not learn these alignments consistently across different language pairs. The reason might be the high difference in length between source and target. The question is usually quite short, but the corresponding query is quite long. Therefore, many words in the target do not really have a corresponding word in the source sequence, e.g. the token "SELECT" is present in every query, but cannot really be matched to a word in the question. A concrete example shows that the "DISTINCT" of the query is sometimes linked to first word of the question and sometimes to the fourth word. Of course, there is not really a correct aligned source word for this query specific word, but consistency might be handy at least.

We made another experiment by using the same input as for C and D, but using a different model architecture, which also trys to learn correct alignments. Unfortunately, no improvement is observed and the values match the previous ones exactly. The reason might be the observed large difference in length be-

Reference	SELECT DISTINCT ?o WHERE {?e eiopa-Base:hasIdentifyingName "5573" . ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o . }
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "? 5573 5573" . ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o . }
Reference	SELECT DISTINCT ?o WHERE {?e eiopa-Base:hasIdentifyingName "generali deutschland versicherung ag" . ?e eiopa-Base:hasNCA ?o . }
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "the the the" . ?e eiopa-Base:hasNCA ?o . }

Table 6.4: Example Translations using Model D

tween source and target or an error in the implementation of the multi-task loss. Further research might be required.

6.3 Model E and F

Model E and F use subwords to handle the correct translation of unknown and rare entity objects. The average BLEU scores on the test sets of 87,65 respective 89,79 indicates that also this set-up is able to generally handle the translation task well. Model E is trained on a dataset where all dutch entity objects are treated as OOV words when training and are accordingly split into subwords. These results in entity mismatches, which we can observe by a lower name match accuracy score of 75,88% for test set 1 and 83,58% for test set 4. Model F is trained on a dataset treating dutch entity objects as known and german as unknown. Accordingly, the dutch entity objects are treated more as rare words, while the german ones are treated as unknown. This setup might be slightly flawed for the subwords models as the german entity objects are actually not completely unknown and the train set can already train on mapping the subowrds of these names correctly. In this dataset, the known entity objects are considered when producing the byte-pair encodings and accordingly will stick together as slightly larger subwords, while the unknowns will be split in more tiny subwords, except they contain consist of subwords used by known entity objects. This leads to a better score in name match accuracy for test set 1 and 4 with 97,26% respective 95,89%. Model E cannot handle the unknown entity names in test set 3 and has 0% accuracy for the name match. Contrary, model F can handle unknown names very well and we can observe a name match accuracy of 83,45% for test set 3 with unknown entity names and insurance id codes.

Reference	SELECT DISTINCT ?o WHERE {?e eiopa-Base:hasIdentifyingName "bos fruit aardappelen onderlinge verzekeringen bfao u.a." . ?e eiopa-Base:hasOperationStartDate ?o . }
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "bos fruit aardappelen onderlinge waarborgmaatschappij u.a." . ?e eiopa-Base:hasOperationStartDate ?o . }
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "roeminck gleif-L1:hasLegalName ?o . ?e eiopa-Base:hasOperationStartDate ?o . }
Reference	SELECT DISTINCT ?o WHERE {?e eiopa-Base:hasIdentifyingName "r140252" . ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o . }
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "r140252" . ?e gleif-Base:identifies ?s . ?s gleif-L1:hasHeadquartersAddress ?a . ?a gleif-base:hasCountry ?o . }

Table 6.5: Example Translations using Model F

As the evaluation metrics show high performance for Model F, we decide to have a closer look at some example translations shown in Table 6.5. These show some issues appearing in the test sets, although not frequently. The first example shows a case of an entity mismatch of an example from test set 4, which overall has a correct match in 97,08% of the cases. In the example at hand, we can observe a small mix-up in the long name "bos fruit aardappelen onderlinge verzekeringen bfao u.a." and the words "verzekeringen bfao" are replaced by "waarborgmaatschappij". Of course due to the nature of neural networks it is rather hard to conclude what is going wrong, but we might argue that likely the embeddings of the two names are close together and maybe the "waarborgmaatschappij" is slightly more often present in the train set. The reason is that unlike in a copy model, the subwords model needs to translate all parts of a sentence via the whole network and the embeddings. Nevertheless, this is rather speculative. An interesting side-effect is that the name "bos fruit aardappelen onderlinge waarborgmaatschappij u.a." does not even exist and the query returns no result. For the second example, we only show the translation of the network. This query in fact produces an error in the process of querying because of a syntactical mistake. The name object at hand is poorly translated with the word "insurance" and the second quotation marks missing. This type of mistake is not frequent and only occurs twice for the 585 questions in test set 1. It is still interesting, how the network is almost always producing syntactical correct queries, but is failing here for a specific question and two specific insurance companies. When looking closer at the results for the other test sets, we observe a couple more erroneous queries, but only when working with then name "roeminck insurance n.v.". Unfortunately, we are not able to find a possible explanation, as we also recheck that the train set is not containing wrong references for this insurance company. In the last example given, one can observe two correct queries with a correct entity translation but the translation not producing the correct result. The reason here is the design of the templates for the train set. We include questions about the operating country and the country of residence. Of course, the questions are quite close together and accordingly in some cases the "wrong" country is returned by the query. In the example, we asked for the operating country, but we get the country where headquarter is located. A possible resolve might be to make the questions less ambiguous, which might be rather difficult as both questions are concerned with some country information for an insurance. This example was taken from test set 1. As the results show an astonishing correctness of 97.59% of the results, we might settle at the fact that in most deep learning settings 100% accuracy is simply impossible.

6.4 Model G and H

The observation made above that it is difficult to learn correct and meaningful embeddings for rarely occurring words, lead us to test the pointer-generator model. The model G and H operate on a word-level basis, but like explained in Section 4.3.5 can decide to copy words directly from the source sentence or generate them from the dictionary. The average BLEU scores over all four test sets are 86,08 for model G and 85,34 for model H. Model G was trained on a dataset where the dutch entity objects are treated as known and the german ones as unknown, whilst model H was only trained on unknowingly treated entity objects. The averaged BLEU scores are slightly lower compared to their equivalent subword models E and F, but still indicate a general high performance. If we further analyse the results for model G, we observe a rather low name match accuracy for test set 1 of 73,29% and also a rather lower accuracy in query results of 73,54%. This is remarkably as the entity objects and queries in test set 1 are known to the model from the train set and only new questions are used.

The analysis of the results indicates a problem with repetition as the examples in Table 6.6 show. Most wrong answers are encountered when asking for information using the insurance id code. The model does not well generalize on the length of the entity object. In the examples at hand, we observe that the model tries to insert two tokens in to the name object while one would be sufficient. Furthermore, we notice the same mismatch with country and operating

Reference	SELECT DISTINCT ?o WHERE {?e eiopa-Base:hasIdentifyingName ”w1943” . ?e eiopa-Base:hasEUCountryWhereEntityOperates ?o . }
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName ”operating w1943” . ?e gleif-Base:identifies ?s . ?s gleif-L1:hasHeadquartersAddress ?a . ?a gleif-base:hasCountry ?o . }
Reference	SELECT DISTINCT ?a WHERE {?e eiopa-Base:hasIdentifyingName ”n.v. levensverzekering-maatschappij de hoop” . ?e eiopa-Base:hasInsuranceUndertakingID ?a . }
Translation	SELECT DISTINCT ?a WHERE{?e eiopa-Base:hasIdentifyingName ”insurance levensverzekering-maatschappij de hoop” . ?e eiopa-Base:hasInsuranceUndertakingID ?a . }

Table 6.6: Example Translations using Model G

country. For this set-up the reason might be that in the unknown question of the test set, the word ”operating” is treated OOV as the vocabulary is rather small and in the train set we used the word ”operate” for this kind of questions. This might explain why the model returns a query on the country of residence. The second example shows also a strange artifact as the word ”insurance” is randomly added to the entity object. See et al. [22] claimed that the problem of repetition is solved by using a coverage loss, but this might not fully work at a word level, although we cannot fully support this conclusion.

6.5 Model I, J, K1 and K2

As the models using subwords and pointer-generator performed better than the other models, we also tested a combined model. However, contrary to the expectations, models I and J perform worse than models models F and G, the best-performing models. The average BLEU score over all 4 test sets is 78,83 for model I and 82,66 for model J, which is slightly worse than the performance of the singular model in terms of BLEU score. Nevertheless, we observe a low score of 7,26% name match accuracy for model I on test set 3, so on unknown names. Model I was trained with dutch entity names, but these were treated not as part of the dictionary. The model is able to handle the dutch names well in test set 4 with an name match accuracy of 95,98%, but the accuracy is only 65,87% for test set 1 with unknown questions. We might conclude that the subwords is not able to counter the inability of a pointer-generator model to generalize well on unknown questions. For Model J, which was trained with partly known and unknown names, the performance is better, but also this model only returns for 53,78% of the questions in test set 1 the correct answer. As the name match accuracy is 72,7% for this test set, we notice a worse translation performance. The first example in Table 6.7 indicates that the model is not able to consistently generate syntactically correct queries and in the example one one of the separation dots is missing between the query triplets. In the second example, the translation misses a part of the name, whilst the another word is present two times.

Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "onderlinge verzekerings maatschappij rijn en aar u.a." . ?e gleif-Base:identifies ?s . ?s gleif-L1:hasLegalForm ?l gleif-base:hasNameLocal ?o . }
Reference	SELECT DISTINCT ?o WHERE {?e eiopa-Base:hasIdentifyingName "zilveren kruis zorgverzekeringen n.v." . ?e eiopa-Base:hasCrossBorderStatus ?o . }
Translation	SELECT DISTINCT ?o WHERE{?e eiopa-Base:hasIdentifyingName "zilveren kruis n.v. n.v." . ?e eiopa-Base:hasCrossBorderStatus ?o . }

Table 6.7: Example Translations using Model J

We decided to make another experiment and test the best performing models on multi-lingual task. As stated before, we translated the questions used in the templates to dutch and train the models on the mixed questions. The setup of model K1 is the same as model F and the setup of model K2 the same as model G. Both models perform well on the general translation task for both dutch and english questions. Model K1 scores an average BLEU score of 80,61 for the english questions in test set 1–4 and 81,01 for the dutch questions. It is interesting to observe that the averaged BLEU score is roughly 9 points lower for the english questions compared to the performance of model F, which was only trained on english questions. There is little difference for the other metrics comparing the english and dutch test sets. The performance on the new questions and queries is remarkable as the model scores much higher in terms of name match accuracy (95,38%) and query result accuracy (48,44%), when evaluated on the dutch questions. This performance is roughly double as high compared to the english questions. On the contrary, the model scores less high on the unknown dutch questions (test set 6) with a query result accuracy of 75,95% compared to 90,71% for the english test set. We are not able to find a concrete reason for this, but we might argue that the translations of the questions from english to dutch might change the difficulty of test sets. A possible explanation might be that we use dutch words in the new questions which are still in the vocabulary and hence the model better understands the questions. Furthermore, we observe the similar struggles in the translation as for model F, referring to entity mismatches and slight omissions or changes to the names of insurances. Model K2 also performs well for the general translation task and has an averaged BLEU score of 81,97 for test sets 1–4 and 82,36 for sets 5–8. Also these scores are lower for the english sets compared to the performance of model G on test sets 1–4. Very remarkable is the performance of model K2 on test set 2, where it scores 0 correct name matches, which is not well explainable to use. We can conclude from the bilingual experiment that it does not necessarily improve the performance of the model and might even perform worse than when only using monolingual data.

6.6 Model L

The set-up with the pretrained XLM-R model did not produce good results at all. In Table 6.1 we give one example. The generated translation is of bad quality. Our relatively small amount of training data might not be enough to properly fine-tune such a large model as the example shows some elements of correct query. Also the model implementation is not correct as the `<mask>` token from pre-training the the masked language model is present in the output.

Listing 6.1: Example Translation for XLMR model

```
r SELECT DISTIN_<mask>lCT WHcke<mask>ERE_<mask> o
ei<mask>dot<mask>baseopenIdentshasbra<mask>quot<mask>mark—<mask>
n17borgancehas bra<mask>quot<mask>Name var<mask> sep_<mask>
ifopa<mask>dot<mask>esa_<mask> gle var<mask> sep_<mask>
gleifopa<mask>markidentifi<mask>baseryLeAddress_<mask>mark
var<mask> sep_<mask>markifopa1dotvbasesForm_<mask>l
var<mask> sepcke<mask>ifying
```

Chapter 7

Conclusion & Further Work

Our research has shown the possibility to use NMT for translating natural language questions to SPARQL queries in the context of a dataset containing data from the EIOPA and GLEIF register. We were able to adapt the approach of Soru et al.[24] for generating the datasets to a new KB and achieve similar performance as reported by Yin et al.[32] for their implementation of a Transformer model. This leads to the conclusion that a NMT is able to produce syntactically and semantically correct queries and is also able to match the correct query to the corresponding question.

However, our results for the standard Transformer architecture (Model A and B) showed shortcomings in translating the entity objects correctly, especially if these were not present in the dictionary and train set, which were also observed in the previous works. We focused our research on the correct translation of the entity objects, since this is important to let the model return the correct result. We test three different approaches to handle the unknown and rare objects and answer the second research question. The first one uses alignments between the tokens in the source and target sequence to replace OOV tokens in the target sequence with the corresponding word in the source sequence. This approach exhibits some improvement, but the observed name match accuracy for unknown names was still rather low. The conclusion here might be that the large difference in length makes it hard to learn correct and consistent alignments between the source and target language. The second approach was to let the models operate at a subword level instead of word level. This approach was successful, showed great strength in translating unknown names and was also able to well handle questions not present in the train set. Nevertheless, as the model is still required to generate names using the encodings and embeddings, entity mismatches are present. Our third approach tried to circumvent that entity objects need to go via the complete model pipeline by using a pointing mechanism. This model architecture has a high name match accuracy for unknown and known names, when tested on known questions (present in the train set). The overall translation performance is rather low when unknown questions are fed to the model. Apparently, the tested model setup is not able to generalize well on new questions. A test to combine the second and third approach unfortunately did not result in a combination of the strengths of each model, but the combination model performed worse than the singular models. To sum up, there is not one single best model and both approaches using subwords and a pointer-generator model show great performance. The trade-off from an user perspective might be whether it is more important to produce correct unknown names in the translation or generalize well for new questions. Depending on the needs, one or the other model should be implemented.

For each model, we tested the performance by training only one with all entity objects as unknown or with a train set containing entity objects treated as part of the dictionary and also treated as unknown, which serves the purpose to answer our third research question. The objective was to see in which setup the model can handle unknown objects better. The train set had the same size in both cases by generating half of the examples for dutch and german insurances compared to only dutch insurances. Surprisingly, the results for all model setups was better when trained with known and unknown names. However, this result might be biased as the performance on unknown names was measured on the german entity objects, which were also used as unknown names. Hence, these names were actually not completely unknown anymore. Still, in subword models these objects were split in many little subwords and in other setups replaced OOV token. Probably, the model was accordingly able to learn how to properly deal with these combinations of subwords or translating OOV tokens. From an user perspective, it might be important to think about what names will be used and include these in the train set, since this improves the match accuracy drastically. Another test could be done in the future by using a set of known and unknown names for training and the using a second unknown names set for testing.

There are more possibilities to extend our research in the future and improve the performance of translating to SPARQL queries. For generating the datasets, we used multiple questions for the same query to ensure some degree of robustness. When analysing the results we noticed that this is not enough and the vocabulary is still quite small, which leads to important words, e.g. "operating", in a new question to be replaced by a OOV token. Even if subwords are used the new word might lead to problems as the encoding algorithm is also only trained on a small vocabulary. Adding a large english dictionary to the source side will not be helpful, as when the words do not occur in the train set, the embeddings of the word will not be trained and consequently the model does not know what this word means. Therefore, another try with a pre-trained mode might be important to improve the language understanding of the model. Furthermore, the templates in our research were limited to questions and queries about one variable and "SELECT" queries. More diversification and different style of templates might be beneficial from an user perspective, who might have more diverse questions than covered at the moment. As the correct translation proved as obstacle for returning the correct results by the query, there exist some possibilities to improve the generation of rare and unknown entity objects. At first, special symbols in names cause problems in the queries due to the structure of SPARQL. We were able to handle this partly by making changed in the KB, but also by adding more replacements for abbreviations when encoding the SPARQL queries. Still, not all abbreviations can be covered and this leads to wrong queries, hence further research should be done here. On the same aspect, we saw that entity mismatches occur, especially when shorter forms of the names are used. A fuzzy name match algorithm might be added to ensure that names are matched in a correct way. This might prove beneficial. Additionally, we only used a non-specific copy network and noticed that sometimes parts of the question are copied to the entity objects which should not be there. A copy network with a class-specific copy algorithm might be beneficial to solve this problem. As last for more advanced research, there might exist several ways in a KB to get the correct result for a question. If the translation quality is evaluated, such a different way might be labelled as incorrect translation although it might be valid to use this query. A reinforcement learning approach might be able to solve this conflict, although we cannot judged how well it would work and how often there exist several queries to get the correct result in a KB.

Bibliography

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

[2] 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.

[3] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

[4] 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.

[5] Miguel Domingo, Mercedes Garcıa-Martınez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. How much does tokenization affect neural machine translation? *arXiv preprint arXiv:1812.08621*, 2018.

[6] Seppo Enarvi, Marilisa Amoia, Miguel Del-Agua Teba, Brian Delaney, Frank Diehl, Stefan Hahn, Kristina Harris, Liam McGrath, Yue Pan, Joel Pinto, et al. Generating medical reports from patient-doctor conversations using sequence-to-sequence models. In *Proceedings of the first workshop on natural language processing for medical conversations*, pages 22–30, 2020.

[7] Sarthak Garg, Stephan Peitz, Udhyakumar Nallasamy, and Matthias Paulik. Jointly learning to align and translate with transformer models. *arXiv preprint arXiv:1909.02074*, 2019.

[8] Michael Grobe. Rdf, jena, sparql and the'semantic web'. In *Proceedings of the 37th annual ACM SIGUCCS fall conference: communication and collaboration*, pages 131–138, 2009.

[9] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. Incorporat-

ing copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

[10] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*, 2016.

[11] Ann-Kathrin Hartmann, Marx E Tommaso, D Moussallem, G Publio, A Valdestilhas, D Esteves, and CB Neto. Generating a large dataset for neural question answering over the dbpedia knowledge base. In *Workshop on Linked Data Management, co-located with the W3C WEBBR*, volume 2018, 2018.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Heikki Hyyrö. Explaining and extending the bit-parallel approximate string matching algorithm of myers. Technical report, Citeseer, 2001.

[14] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180, 2007.

[15] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

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

[17] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

[18] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[19] Anand Panchbhai, Tommaso Soru, and Edgard Marx. Exploring sequence-to-sequence models for sparql pattern composition. In *Iberoamerican Knowledge Graphs and Semantic Web Conference*, pages 158–165. Springer, 2020.

[20] 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, 2002.

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

[22] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[23] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[24] Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, André Valdestilhas, Diego Esteves, and Ciro Baron Neto. Sparql as a foreign language. *arXiv preprint arXiv:1708.07624*, 2017.

[25] Tommaso Soru, Edgard Marx, André Valdestilhas, Diego Esteves, Diego Moussallem, and Gustavo Publio. Neural machine translation for query construction and composition. *arXiv preprint arXiv:1806.10478*, 2018.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[27] Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien Nguyen. Does bleu score work for code migration? In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 165–176. IEEE, 2019.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[29] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

[30] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[31] Shuoheng Yang, Yuxin Wang, and Xiaowen Chu. A survey of deep learning techniques for neural machine translation. *arXiv preprint arXiv:2002.07526*, 2020.

[32] Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural machine translating from natural language to sparql. *Future Generation Computer Systems*, 117:510–519, 2021.