TRIVIANLQ: NATURAL LANGUAGE QUERY FOR TRIVIAL QUESTIONS

UE905 SOFTWARE PROJECT

Karolin Boczoń
IDMC
University of Lorraine
Nancy, France
boczon1u@etu.univ-lorraine.fr

Dimitra Niaouri
IDMC
University of Lorraine
Nancy, France
niaouri9u@etu.univ-lorraine.fr

Roham Roshanfekr
IDMC
University of Lorraine
Nancy, France
roshanfe1u@etu.univ-lorraine.fr

Muhammad Shahzaib
IDMC
University of Lorraine
Nancy, France
shahzaib1u@etu.univ-lorraine.fr

ABSTRACT

This paper introduces TriviaNLQ [1], a web-application that utilizes text-to-SPARQL modeling to query the DBpedia knowledge graph and answer trivia questions. It also acts as a survey on available text-to-SPRARQL models. For building our application we trained a Convolutional Sequence-to-Sequence model (CNN architecture) based on a Neural Machine Translation(NMT) approach using Facebook's Fairseq toolkit. The datasets that we used are LC-QUAD and Monument datasets. Our results suggest a better performance of the model when using the Monument dataset, having a BLEU score of 97.6 on the validation set and 97.3 on the test set.

1 Introduction

Knowledge graphs - also known as semantic networks - are an effective concept for representing real-world entities and the relationships between them. These graphs are typically massive and frequently inaccessible to users since accessing those requires specialized knowledge in query languages such as SPARQL [Liang et al., 2021]. Expertise in the language's syntax and semantics is highly demanding and could not be assumed to be possessed by average human web users [Yin et al., 2019]. Furthermore, users need to have a good understanding of the structure of the underlying data models, which are normally based on the Resource Description Framework (RDF). These disadvantages have prompted the development of Question-Answering (QA) systems, which allow users to access those knowledge graphs using natural language [Liang et al., 2021]. While existing systems make it easier for users to access information, their accuracy could be improved.

Interested in text-to-SPARQL modeling, in this study, we aim to develop an application that answers trivial questions. Given the time constraint on the project, instead of creating our own model, we wish to use an existing one. In this paper we will compare several text-to-SPARQL models in order to determine which model performs the best in terms of accuracy, efficiency, and scalability. This comparison will allow us to identify the strengths and weaknesses of each model and make informed decisions about which model to use for TriviaNLQ. Additionally, we will also evaluate the models on their ability to handle a wide range of trivia questions, from simple factual questions to more complex, multi-fact questions, to ensure that TriviaNLQ can provide accurate answers to a diverse set of queries. Overall, this comparison will provide valuable insight into the capabilities of different text-to-SPARQL models and help us achieve good performance of TriviaNLQ.

[1] https://github.com/lyrs/TriviaNLQ

The final product we wish to develop will allow users to ask questions in natural language and receive answers without having to query knowledge bases using the structured query language (SPARQL). This is beneficial for a number of reasons. Firstly, it allows users to access information from knowledge bases without having to learn the syntax of SPARQL. Secondly, it enables them to ask more complex questions than they would be able to with a simple keyword search. Finally, it allows them to access information from multiple knowledge bases at once, making it easier to find the answers they are looking for. In addition, to our knowledge, no such system has been developed for addressing trivia questions. We find this domain of particular interest as trivial questions require a broad range of knowledge and the ability to retrieve specific, often obscure, information. Given that using a knowledge graph such as Wikidata allows us to access a vast amount of structured data, we decided to dedicate this project to developing an accurate system that allows users to easily get answers to trivia questions. As trivia questions are an important part of learning and understanding the world around us, such a system could prove to be a useful tool for users to gain knowledge about a variety of topics, from history and science to pop culture and current events. However, apart from the informative and cultural aspects of such a trivia question/answering system, we should not disregard the entertaining ones. Specifically, this system could prove to be a fun and engaging tool for users, allowing them to connect with others over trivia games, as they provide an opportunity for people to share their knowledge and learn from one another. Additionally, trivia questions can help to stimulate conversation and debate, as people may disagree on the correct answer or have different opinions on a particular topic.

The remainder of this study is structured as follows: in Section 2 we present the theoretical background, then in Section 3 we talk about our methodology. In Section 4 we present our results that discuss them in detail in Section 5. Finally, we show the limitations of the project and possible solutions in Section 6.

2 Theoretical Background

The theoretical background section of this report aims to provide an overview of the fundamental concepts and techniques that are used in text-to-SPARQL modeling. It covers the basic principles of knowledge representation, semantic parsing and neural language models, which are essential for understanding how text-to-SPARQL models work.

2.1 Knowledge graphs in comparison to knowledge bases

As presented by Cook [2020] knowledge bases are collections of structured or unstructured data that are used within an information system. The term was initially created to separate databases, which contained flat, tabular data, from the newer knowledge bases with relational and hierarchical data. Nowadays, databases are as sophisticated as the knowledge bases of the past. The implication is that current knowledge bases are closer to containing semantically and organizationally multifaceted data. While some researchers may use the terms knowledge base and knowledge graph interchangeably, there are differences between them. All knowledge graphs are knowledge bases, whereas not every knowledge base is a knowledge graph. The major distinction between knowledge graphs and bases is that graphs are focused on the connections between entities. Graphs are also both mutable and semantic, which means that the meaning of the data is encoded together with the data. This provides great potential for machine learning in automated knowledge bases or knowledge graphs [Cook, 2020]. An example of a knowledge graph is presented in Figure 1.

2.2 Ontologies

Ontologies provide the backbone of a knowledge graph's formal semantics, serving as the data schema for the graph [Fundamentals, 2022]. Since the 1990s, ontologies have been the subject of research in the AI field, and are now being utilized in many other areas. These include natural language processing, cooperative information systems, information retrieval, electronic commerce, and knowledge management [Fensel, 2001]. The main reason for the growing popularity of ontologies is the potential they offer for knowledge sharing and reuse between people and applications [Fensel, 2001]. They ensure a consistent understanding of the data and its meaning by the developers and users, whether that user is a person or a software application [Fundamentals, 2022]. Ontologies employ a number of representation and modeling instruments, including classes, relationship types, categories, and free text descriptions [Fundamentals, 2022]. Classes provide a classification of entities with respect to a class hierarchy and relationship types detail the nature of the relationship between entities [Fundamentals, 2022]. Categories help to describe and order entities into a taxonomy, while free text descriptions can provide a more human-friendly explanation of design intentions [Fundamentals, 2022].

2.3 RDF

The Resource Description Framework (RDF) is a data model used to represent information about resources on the Web, which is recommended by the World Wide Web Consortium (W3C) [Angles and Gutierrez, 2008]. The RDF specification contains a set of predefined keywords, known as the RDFS vocabulary, which are used to describe relationships between resources, such as typing and inheritance of classes and properties [Angles and Gutierrez, 2008].

RDF is based on the concept of an object-attribute-value triple, which can be expressed as A(O,V) [Decker et al., 2000]. This can also be visualized as a labeled edge, A, between two nodes, O and V: [O]–A→[V] [Decker et al., 2000]. This notation is advantageous because RDF allows for objects and values to be interchangeable, thus allowing two labeled edges to be chained in a graphic representation[Decker et al., 2000]. An example of a graph of RDF triples is presented in Figure 2.

2.4 SPARQL

SPARQL is the standard language for querying RDF data. SPARQL queries consist of three parts: pattern matching, solution modifiers, and output [Pérez et al., 2009]. Pattern matching includes several features such as optional parts, union of patterns, nesting, filtering values, and the ability to choose the data source [Pérez et al., 2009]. Solution modifiers allow the output of the pattern to be modified by applying operators such as distinct, order, and limit [Pérez et al., 2009]. Finally, the output of a SPARQL query can be in the form of yes/no queries, selections of values that match the patterns, construction of new RDF data, or descriptions of resources [Pérez et al., 2009]. An example of a simple SPARQL query is presented in Figure 3.

As we can see from the depicted example, variables in SPARQL start with a ? and can match any node, either a resource or literal, in the RDF graph. Triple patterns are similar to triples, except that any of the parts of a triple can be substituted with a variable. The SELECT result clause returns a table of variables and values that fulfill the query. In other words, the query finds all the names mentioned in Tim Berners-Lee's FOAF file by locating all subjects (?person) and objects

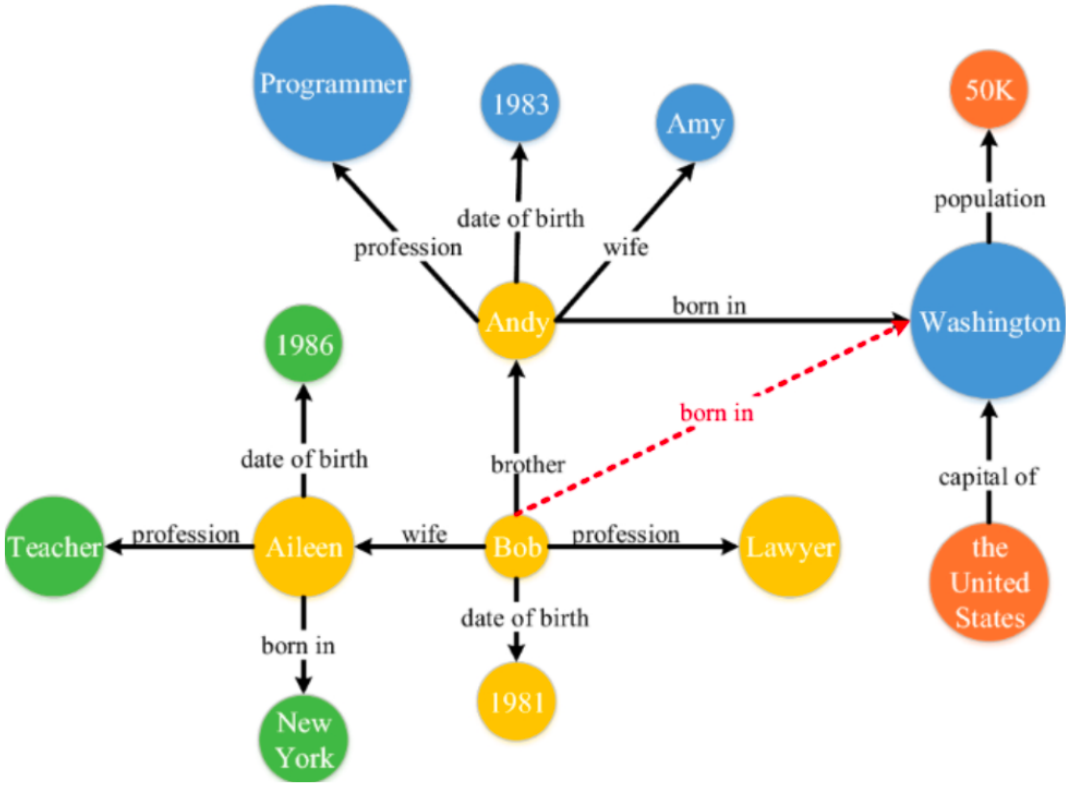

Figure 1: Example of a Knowledge graph [Kumar, 2022]

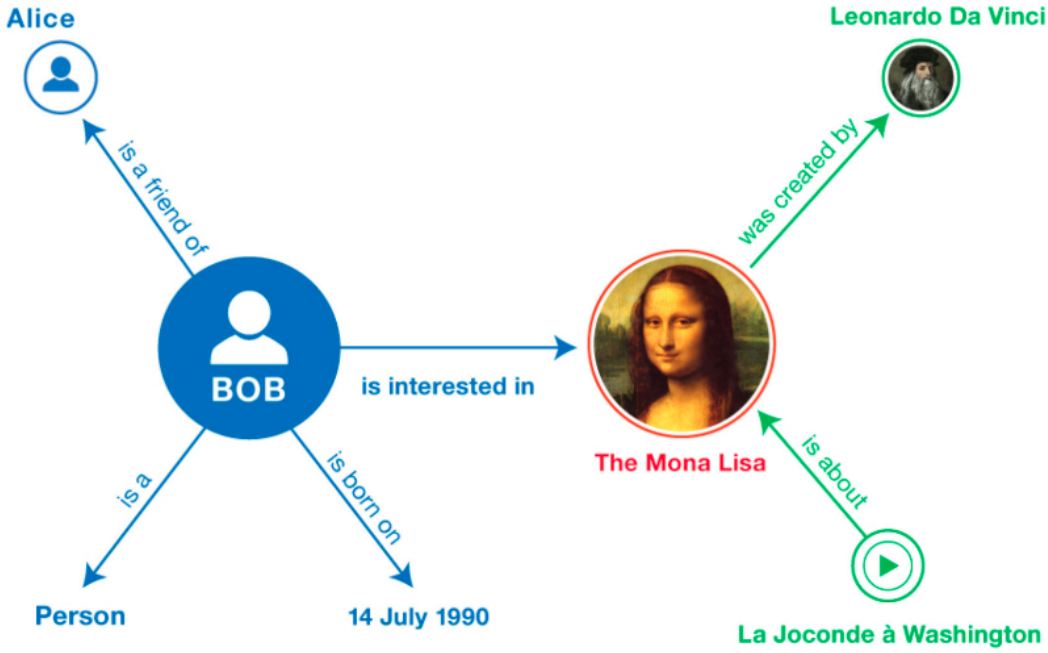

Figure 2: Example of a graph of RDF triples [Wylot et al., 2018]

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
    ?person foaf:name ?name .
}
```

Figure 3: Example of a simple SPARQL query[Feigenbaum, 2009]

(?name) linked with the foaf:name predicate in the graph http://www.w3.org/People/Berners-Lee/card, and then returns all the values of ?name [Feigenbaum, 2009].

2.5 Neural Machine Translation

As presented in the study of Tan et al. [2020] Machine Translation (MT) is a crucial task that aims to translate natural language sentences through computer-based technology. Initially, the approach to machine translation heavily relied on the use of manually crafted translation rules and linguistic knowledge [Brown et al., 1990, Koehn, 2003]. However, due to the complexity of natural languages, it was challenging to encompass all language irregularities through manual translation rules [Tan et al., 2020]. For this reason, new technologies in the domain of MT had to arise.

With the advancement of deep learning, Neural Machine Translation (NMT) has emerged as a new paradigm that uses artificial neural networks to translate text from one language to another [Kalchbrenner and Blunsom, 2013, Cho and van Merriënboer, 2014]. Being trained on large parallel corpora of text in the source and target languages, where the model learns to translate text by predicting the probability of the next word in a sentence, NMT models are considered to be more advanced than traditional rule-based and statistical machine translation methods as they can model the entire context of a sentence, rather than breaking it down into smaller phrases or words or learning latent structures such as word alignments or phrases directly from parallel corpora [Junczys-Dowmunt et al., 2016]

Tan et al. [2020] explain that NMT utilizes a single large neural network to model the entire translation process, eliminating the need for extensive feature engineering and achieving state-of-the-art performance on various language pairs. Specifically, it uses a neural network architecture known as an encoder-decoder architecture. The encoder network processes the source text and creates a representation of the sentence, and the decoder network generates the target text by using this representation. The main difference between SMT and NMT is that NMT employs continuous

representations as opposed to discrete symbolic representations utilized by SMT and its training process is end-to-end. Furthermore, NMT has the ability to model long-distance dependencies between words which improves the quality of the translation. Finally, it is trained end-to-end to learn the mapping between the source and target languages, which allows the model to learn to translate not only individual words but also idiomatic phrases and expressions [Tan et al., 2020].

2.5.1 NMT and text-to-SPARQL

Neural Machine Translation can be used to translate natural language sentences (such as English text) into a structured query language, such as SPARQL as extensively demonstrated in the study of Yin et al. [2019]. The process involves training an NMT model on a large dataset of aligned text-SPARQL pairs. During inference, a natural language sentence is input into the trained model, which then generates a corresponding SPARQL query. This approach has been used for various natural language processing tasks, including knowledge base querying, semantic parsing, and information retrieval. However, one of the main challenges with using NMT for text-to-SPARQL is the limited availability of large, high-quality parallel data.

In a study conducted by Yin et al. [2019], the effectiveness of various neural network architectures, including RNN, CNN, and Transformer, were evaluated in their ability to translate natural language into SPARQL queries. In their study, they found that the ConvS2S model significantly outperformed all other models, with a BLEU score of up to 98% and an accuracy of up to 94% (for more details see 2.6).

Having one of the best performances in the literature for translating text to SPARQL, the architecture of a ConvS2S model introduced by Gehring et al. [2017a] is further implemented in this study. For this reason, a detailed presentation of the Sequence to Sequence learning technique and the architecture of Convolutional Neural Networks will follow in the next sections.

2.5.2 Sequence to Sequence Model

Sequence to Sequence (Seq2Seq) learning is a type of machine learning technique where the model is trained to map input sequences to output sequences. As presented in the study of Gehring et al. [2017a] sequence-to-sequence learning has been a widely adopted approach for a range of natural language processing tasks such as machine translation, speech recognition, and text summarization [Sutskever et al., 2014, Chorowski et al., 2015, Rush et al., 2015, Nallapati et al., 2016, Shen et al., 2016]

The task of a Seq2Seq modeling is to generate a target sequence from a given seed sequence [Li et al., 2018]. The model is typically composed of two parts: an encoder that encodes the seed sequence into a hidden variable and a decoder that generates the target sequence from the hidden variable [Li et al., 2018]. The encoder-decoder architecture is commonly used in Seq2Seq learning for automated translation. An example of a conventional encoder-decoder architecture for machine translation architecture is presented in the figure below. As explained in the study of Yin et al. [2019], the encoder takes in a sequence of tokens as input and converts it into a feature vector that represents the input information in a higher dimensional space. The decoder then takes this feature vector and generates a new sequence of tokens as output. This architecture has been shown to be superior to traditional phrase-based models because it is easier to extract features, is more flexible in terms of model configuration, and leads to better results in terms of accuracy[Yin et al., 2019].

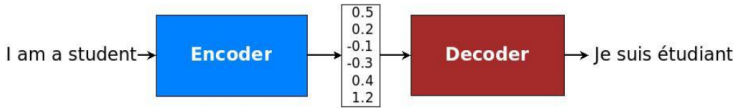

Figure 4: A conventional encoder-decoder architecture for machine translation [Luong et al., 2017, Yin et al., 2019]

An attention mechanism is a commonly used component in Seq2Seq learning, particularly in machine translation and computer vision tasks [Yin et al., 2019]. The attention mechanism connects the decoder with the encoder's hidden states, allowing the decoder to selectively focus on specific parts of the input, rather than relying solely on a global representation [Yin et al., 2019]. As a result, the use of the attention mechanism improves the performance of NMT approaches by addressing the long-term dependency problem.[Yin et al., 2019].

While Recurrent Neural Networks (RNNs) are often used for sequential data, convolutional models have also been applied to sequence-to-sequence tasks, such as machine translation [Li et al., 2018]. Their effectiveness lies in their inherent advantage in computation parallelization and their ability to build hierarchical representations [Yin et al., 2019]. In the following section the architecture of CNN models will be presented in detail.

2.5.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have been dominating the field of image processing and have been less frequently employed in comparison to Recurrent Neural Networks (RNNs) for machine translation tasks. Despite this, CNNs have been demonstrated to possess several advantages, such as improved feature extraction capabilities and faster training speed [Yin et al., 2019]. As elaborated in the study of Gehring et al. [2017a], CNNs create representations for fixed-length contexts and the effective context size of the network can be easily increased by stacking multiple layers. This allows for precise control of the maximum length of dependencies to be modeled. Additionally, CNNs do not depend on the computations of previous time steps, enabling parallel computation within a sequence. This contrasts with RNNs, which maintain a hidden state of the entire past, preventing parallel computation. Multi-layer CNNs create hierarchical representations over the input sequence, where nearby input elements interact at lower layers and distant elements interact at higher layers. This hierarchical structure provides a more efficient path to capture long-range dependencies compared to the chain structure modeled by RNNs [Gehring et al., 2017a].

In the study of Yin et al. [2019], a specific NMT model referred to as Convolutional Sequence to Sequence model is presented, in which multiple convolutional layers are applied to the input and target sequences in order to extract hierarchical representations. As explained in their thesis, the convolutional operations utilized in natural language processing are typically one-dimensional, as opposed to the two-dimensional operations commonly employed in image processing. An example of a one-dimensional convolutional layer is presented in the image below, in which a kernel of size k = 3 maps a concatenation of k continuous input elements to a single output element [Yin et al., 2019]. The output is computed by applying a weight matrix, bias, and non-linearity Yin et al. [2019]. In order to build deep hierarchical representations through the stacking of CNN layers, padding is also necessary [Yin et al., 2019].

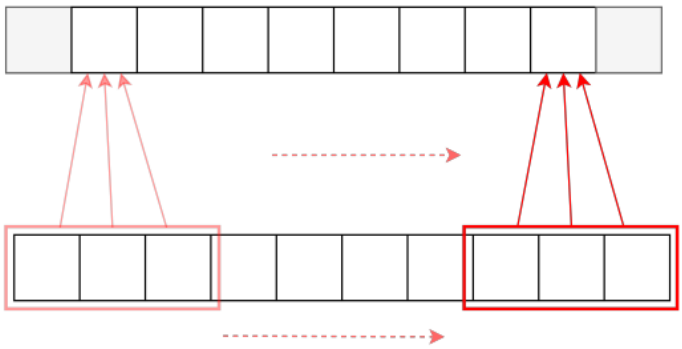

Figure 5: Illustration of a one-dimensional convolutional layer [Yin et al., 2019]

The illustration depicts the input and output embeddings represented by white squares [Yin et al., 2019]. The convolution kernel, represented by the red rectangle, scans over the sequence as indicated by the dashed arrows [Yin et al., 2019]. The grey boxes placed in front of the output sequence represent the padding, which are typically implemented as zero vectors [Yin et al., 2019].

2.6 Related Work

Several studies have addressed the issue of translating Natural Language (NL) Text into a SPARQL Query aiming to make easier the process of querying knowledge graphs of RDF triples such as Wikidata and DBpedia. Soru et al. [2017, 2018] proposed "Neural SPARQL Machines (NSpM)", a generator-learner-interpreter architecture for translating any NL expression to encoded forms of SPARQL queries. They created templates with variables that could be filled with instances from specific types of concepts in the target knowledge base and then generated NL and SPARQL query pairs. After encoding the original SPARQL queries' operators, brackets, and URIs, the pairs were fed into a sequence-to-sequence learner model as training data. The model produced SPARQL encoding sequences for the interpreter to decode, and generalized to previously unknown NL sentences.

Luz and Finger [2018] used an LSTM encoder-decoder model to encode natural language and decode it into SPARQL. They implemented a neural probabilistic language model to learn a SPARQL word vector representation and an attention mechanism to associate a vocabulary mapping between NL and SPARQL. For the implementation of their experiment, they converted the logical queries in the Geo880 dataset into the equivalent SPARQL form. They used two metrics for evaluation: accuracy and syntactic errors. While they obtained logical results, they did not address the out-of-vocabulary issue or lexical ambiguities arising.

Another approach toward the creation of a text-to-SPARQL model comes from the Lymba project [2]. This model was created for building a knowledge base from text, as well as converting text to SPARQL. Specifically, during the first step of the model's implementation, a query in NL passes through the Lymba pipeline and the system creates a semantic representation of the data. Then the system transforms the plain NL question into SPARQL, makes a database query, and displays the information retrieved. Despite the efficiency of the model, the architecture of this model is not publicly available and only serves commercial purposes and not research or academic ones.

Text-to-SPARQL has also been included in Python, as a framework named Quepy for converting natural language questions into database query language queries [Machinalis, 2018]. According to its documentation, it is easily adaptable to various types of natural language and database queries, supporting the query languages Sparql and MQL. However, a big drawback of this framework is the fact that it is written in Python 2, and thus is less convenient to use.

In a comparative study of Yin et al. [2019], eight models from three representative neural network architectures (RNN, CNN, and Transformer) were tested to see how well they could automatically translate SPARQL queries from natural language. Each of these models was trained and tested on three distinct datasets and evaluated using perplexity, BLEU scores, and an accuracy measure. The results of the study suggested that the ConvS2S model significantly outperformed all other models by a wide margin having a BLEU score of up to 98% and an accuracy of up to 94%.

Spiegel et al. [2020] in their study, aimed to automatically generate synthetic English/SPARQL datasets through the generation and refinement of question/query templates that could be used by translation systems to convert NL questions into queries. By developing a basic baseline model for an English-to-SPARQL machine translation task, they were able to successfully assess the usefulness of their dataset. The effectiveness of their approach lies in the fact that they proved that QA systems can be trained without crowdsourcing query/question data. However, despite the benefits of this approach, the need for an initial configuration of the queries provided reduces the automaticity of the model, rendering its implementation more demanding.

Another text-to-SPARQL approach comes from Yazdipour [2021] who implemented a Finetuned T5 model for Text-to-SPARQL conversion. Several different versions of the same model are available, yet all the results were not satisfactory (e.g. F1 score 33% for the text-to-sparql-t5-base).

Finally, Saparina and Osokin [2021] introduced a different approach to converting NL to SPARQL queries. Their pipeline is made up of two components: a non-trainable SPARQL transpiler and a neural semantic parser that transforms natural language queries into intermediate representations. As SPARQL's queries are structurally more similar to their intermediate representations, they chose it over SQL. Their results suggest that the execution accuracy of the SQL queries built by their model on the difficult Spider dataset is comparable to that of the most advanced text-to-SQL techniques developed using annotated SQL queries (For additional information on each of the models see 3).

3 Methodology

This section includes details on the methodological steps followed in this study. First, the models that we tested before the final selection of the compatible model for our study will be presented along with the difficulties that we faced during their implementation. Next, we will focus on our dataset, the preprocessing steps that we followed, and the experimental setup of the CNN-based Convolutional Sequence-to-Sequence model we used.

3.1 Initial testing of available models

In order to select the most appropriate model for our project we had to run a variety of available models and decide on which one fits best the purposes of our project. In the sections below the models we tested will be presented in detail. For additional imformation for each model see also 2.6

3.1.1 Fine-tuned T5 model for Text-to-SPARQL conversion

The first model that we run was the model of Yazdipour [2021] who implemented a Fine-tuned T5 model for Text-to-SPARQL conversion. While testing the model we noticed grammatical and syntactical mistakes in the questions used and most of the output queries were not giving a result when run on the DBpedia or Wikidata endpoint. An example query, along with its result is presented in figure 6

We can notice that: (a) square brackets in the results section have to be replaced with curly ones, (b) the target question used, matches more than one result, and not necessarily female actresses, (c) the output query is not working. Additionally, as already mentioned in 2.6, the model had a low F1 score for all the different available versions of it

[2] https://www.lymba.com/

QUESTION	which female actress is the voice over on south park and is employed as a singer?
Target	SELECT ?answer WHERE { wd:Q16538 wdt:P725 ?answer . ?answer wdt:P106 wd:Q177220}
RESULT	select distinct ?sbj where [?sbj wdt:voice_over wd:south_park . ?sbj wdt:instance_of wd:female_actress]

Figure 6: Example query and result for the text-to-sparql-t5-small-quald9 model

(e.g. 22% for the text-to-sparql-t5-small-quald9), so we decided to not proceed with it as our main model. The training results of the text-to-sparql-t5-small-qald9 model are presented in Figure 7.

Training Loss	Epoch	Step	Validation Loss	Gen Len	P	R	F1	Bleu-score	Bleu-precisions	Bleu-bp
No log	1.0	51	2.4477	19.0	0.3797	0.0727	0.2219	9.3495	[73.47751849743882, 49.595519601742375, 35.346602608098834, 26.243305279265492]	0.2180

Figure 7: Training results of text-to-sparql-t5-small-qald9 [Yazdipour, 2021]

3.1.2 Synthesizing Questions using Iterative Template-Filling (MK-SQUIT)

The next model that we considered was the one of Spiegel et al. [2020]. The authors of the study use WikiData as a source for creating questions and queries. They provide a sample set of 110,000 question-answer pairs from four different areas of WikiData and demonstrate a basic model that performs well in a practical question-answering scenario using this data.

When running the model we noticed that the queries are not complete and thus it was needed to take care of the NEs and match them according to an RDF graph we would have to use.

Question Type	Question	Query
Single-entity	Who is the mother of the director of Pulp Fiction?	SELECT ?end WHERE { [Pulp Fiction] wdt:P5 / wdt:P25 ?end . }
Multi-entity	Is John Steinbeck the author of Green Eggs and Ham?	ASK { BIND ([John Steinbeck] as ?end) . [Green Eggs and Ham] wdt:P50 ?end . }
Count	How many awards does the producer of Fast and Furious have?	SELECT (COUNT (DISTINCT ?end) as ?endcount) WHERE { [Fast and Furious] wdt:P162 / wdt:P166 ?end . }

Figure 8: Example of a Question-Query pair in the study of Spiegel et al. [2020]

However, it needs to be noted that this model has a better performance than the previous one (0.98 and 0.59 BLEU scores for the two datasets tested(test-easy, test-hard respectively)). In figure 9, the results of the model of Spiegel et al. [2020] are presented.

Despite the benefit of this approach, the need for an initial configuration of the queries provided reduced the automaticity of the model, rendering its implementation more demanding.

Dataset	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-W
TEST-EASY	0.98841	0.99581	0.99167	0.99581	0.71521
TEST-HARD	0.59669	0.78746	0.73099	0.78164	0.48497

Figure 9: Training results of the model of Spiegel et al. [2020]

3.1.3 Quepy Python Framework and Querying ontologies

We also tried to implement the Python framework "Quepy" for converting natural language questions into database query language queries [Machinalis, 2018] and a project named "Wine Ontology" [3] which contains a model to read a natural language input and generate a SPARQL query to query the wine ontology. However, both were written in Python 2 and their reimplementation was very difficult. Additionally, as we decided to base our project on a less outdated framework, those two approaches could not serve the purposes of this project.

3.1.4 LYMBA

Lymba [4] project is designed to create a knowledge base from text and convert text into SPARQL, a query language for databases, for use by a wide audience. When a NL question is submitted, it is processed through the Lymba pipeline, which creates a semantic representation of the data. The system then converts the plain language query into SPARQL and uses it to retrieve information from the database. The creation of the knowledge base is done using the Lymba K-Extractor™, which includes tools for identifying 86 different types of entities and 26 semantic relationships, and an ontology is used to understand the context of the search.

We contacted the company behind the Lymba project in order to request some information that would allow us to run their model, yet the fact that the model is not publicly available and only serves commercial purposes led us to investigate other alternative options.

3.1.5 Sparqling Queries

Next, we tested the model of Saparina and Osokin [2021]. The authors presented an interesting approach in which they use an intermediate representation called Question Decomposition Meaning Representation (QDMR). However, the project was developed with SQL and database querying in mind. Despite the promising results of their study, we could not transfer their approach onto our domain. Apart from broken dependencies, the infrastructure of the project was too complicated and we did not succeed in running the model.

3.1.6 GPT3 for Text-to-SPARQL conversion

GPT3 [5] was also tested to see whether it could serve our project. Despite the fact that the model most of the time provides SPARQL queries that seem well structured, in fact, they don't execute due to syntactic errors and don't match objects and predicates. An example of a query structured by GPT3 and executed on DBpedia's endpoint is presented in Figure 10. The output of the query is a syntactic error.

3.2 TriviaNLQ

Our approach, TriviaNLQ [6], is based on the study of Yin et al. [2019] who conducted a comparative study on eight models from three representative neural network architectures (RNN, CNN, and Transformer). We specifically chose to use the model that offered the best results for the task of translating SPARQL queries from natural language, which was a CNN-based Convolutional Sequence-to-Sequence model. The implementation of this model will be presented extensively in the following sections.

[3] https://github.com/ThirasaraAriyarathna/WineOntology
[4] https://www.lymba.com/
[5] https://openai.com/api/
[6] https://github.com/lyrs/TriviaNLQ

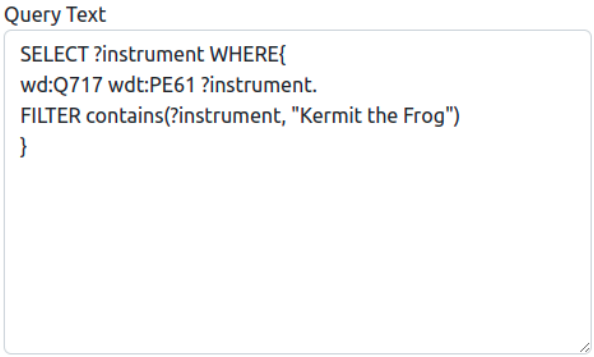

Figure 10: Example query formed by GPT3 executed on DBpedia's endpoint

3.2.1 Datasets

As noted by Yin et al. [2019] the construction of a dataset for training a neural machine translation (NMT) model for translating natural language text into SPARQL queries presents several challenges, including the need for expertise in SPARQL and knowledge of corresponding knowledge bases. The most common method for constructing such a dataset is to manually create a list of template pairs with placeholders and then replace them with entities or predicates from an online knowledge base. However, this method is limited by the complexity of SPARQL and the fact that the vocabularies of online knowledge bases can change over time. As a result, the datasets used in this field typically include only a subset of SPARQL operators (SELECT, ASK, DISTINCT, WHERE, FILTER, ORDER BY, LIMIT, GROUP BY, UNION) [Yin et al., 2019].

LC-QUAD For implementing the CNN model we tested two datasets. The first one is the Largescale Complex Question Answering Dataset (LC-QUAD) is a dataset that contains 5,000 pairs of English questions and their corresponding SPARQL queries [Yin et al., 2019]. As mentioned in the study of Yin et al. [2019], the dataset is designed to provide a large dataset of complex questions, where the complexity of a question is determined by the number of triples in its intended SPARQL query. The dataset uses 38 unique templates and involves the use of 5,042 entities and 615 predicates from DBpedia in its generation workflow. The data generation process in LC-QUAD is different from other datasets, where an entity seed list and a predicate whitelist are prepared in advance, and then subgraphs are extracted from DBpedia through a generic SPARQL query. The triples in the subgraphs are used to instantiate the SPARQL and English templates, which are then reviewed for grammatical correctness [Yin et al., 2019].

Template	SELECT ?uri WHERE { ?x e_in_to_e_in_out e_in_out . ?x e_in_to_e ?uri . }
Query	SELECT ?uri WHERE { ?x dbp:league dbr:Turkish_Handball_Super_League . ?x dbp:mascot ?uri . }
NNQT Instance	What is the <mascot> of the <handball team> whose <league> is <Turkish Handball Super League >?
Question	What are the mascots of the teams participating in the turkish handball super league?

Figure 11: Example question and corresponding instantiation of the query template and the Normalized Natural Question Templates (NNQT) in LC-QUAD generation [Yin et al., 2019]

It is worth noting that the LC-QUAD dataset provides a wider range of questions compared to the Monument dataset, as the Monument dataset is specified to the domain of monuments. The LC-QUAD dataset, while providing complex questions, is limited in size, making it difficult to train deep neural network models on [Yin et al., 2019]. Additionally, while peer reviews were used to check the accuracy of the generated questions and queries, there are still grammar errors present and some questions contain unusual punctuation that can lead to undetected errors in tokenization during vocabulary building [Yin et al., 2019]. For this reason, we decided to not base our model on this dataset, yet try another dataset with a bigger collection of question-query pairs.

Monument The second dataset we used for training was Monument dataset as proposed by Soru et al. [2017]. It comprises a collection of 14,788 question-query pairs in English and SPARQL, respectively. These pairs were generated by utilizing 38 manually crafted template pairs and related assistant SPARQL queries that can be executed directly on the DBpedia endpoint [Hirigoyen et al., 2022, Yin et al., 2019]. The data is restricted to the instances of a specific class dbo:Monument, with a vocabulary size of about 2,500 for English and 2,200 for SPARQL [Yin et al., 2019]. The dataset is generated by filling the placeholders in the templates with the entities and labels retrieved from the DBpedia endpoint by executing the assistant query [Yin et al., 2019]. An example of such a template pair in the Monument dataset is presented in Figure 13.

Question template	Query template

Where is <*A*> ?	`SELECT ?x`
	`WHERE`
	`{`
	`  <A> dbo:location ?x .`
	`}`

Figure 12: Template pair in the monument dataset [Yin et al., 2019]

In order to obtain a list of entities and their corresponding English labels we had to replace placeholders within template pairs by executing a specific SPARQL query on a DBpedia endpoint. The query includes two primary operations, the imposition of a class restriction and the expression of the meaning of the template question through the use of specific triples [Yin et al., 2019]. The resulting values of `?uri` and `?label` are then utilized in pairs to substitute the placeholders in the templates [Yin et al., 2019]. An example of such an assistant query to retrieve a list of entities and labels to fill a template pair is presented in Figure 13.

```
SELECT ?uri ?label
WHERE
{
  ?uri rdf:type <C> .
  ?uri dbo:location ?x .
  ?uri rdfs:label ?label .
  FILTER(lang(?label) = 'en') .
}
```

Figure 13: Example of an assistant query to retrieve a list of entities and labels [Yin et al., 2019]

3.2.2 Preprocessing

The process of training a sequence-to-sequence model for SPARQL involves converting the SPARQL query into a sequence of short tokens [Yin et al., 2019]. This sequence serves as the input for the model, allowing it to learn how to translate natural language text into the encoded form of SPARQL [Yin et al., 2019]. Below details on the SPARQL encoding followed in this study are presented.

SPARQL Encoding With respect to the SPARQL encoding we followed the approach of Yin et al. [2019] and Soru et al. [2017] and converted the structured SPARQL queries into a sequence format compatible with our sequence-to-sequence model. In order to do so, we applied various operations such as shortening entities with prefixes, replacing built-in symbols with dedicated words connected with underscores, and simplifying built-in set phrases. These operations were implemented as a set of replacements, resulting in a final sequence of tokens that were composed solely of characters and underscores. Once the model was trained, the encoded SPARQL query could be easily decoded by reversing the applied replacements [Yin et al., 2019]. An example of such an operation is presented in the Figure 14

SPARQL	SELECT DISTINCT ?uri
	WHERE {
	<http://dbpedia.org/resource/Sam_Loyd> <http://dbpedia.org/ontology/knownFor> ?uri .
	<http://dbpedia.org/resource/Eric_Schiller> <http://dbpedia.org/ontology/knownFor> ?uri .
	}

Encoded	select distinct var_uri where brack_open dbr_Sam_Loyd dbo_knownFor var_uri sep_dot dbr_Eric_Schiller dbo_knownFor var_uri sep_dot brack_close

Figure 14: Example of a converted SPARQL query into an encoded form compatible with the input format required by seq2seq models [Yin et al., 2019]

3.2.3 Convolutional Sequence-to-Sequence model

The model that we trained is a Convolutional Sequence-to-Sequence model which was introduced by Gehring et al. [2017a] and was initially trained for the translation of English to German sentences. In figure 15, the architecture of the Convolutional Sequence-to-Sequence model is presented. This ConvS2S model is a variation of the encoder-decoder architecture, where the encoder and decoder are composed of multiple stacked convolutional blocks [Yin et al., 2019]. Each block is composed of a one-dimensional convolutional layer and a Gated Linear Unit (GLU) as the non-linearity, with a residual connection between the input and output of the GLU [Yin et al., 2019]. Additionally, the decoder also utilizes an attention mechanism, similar to RNN-based models [Yin et al., 2019].

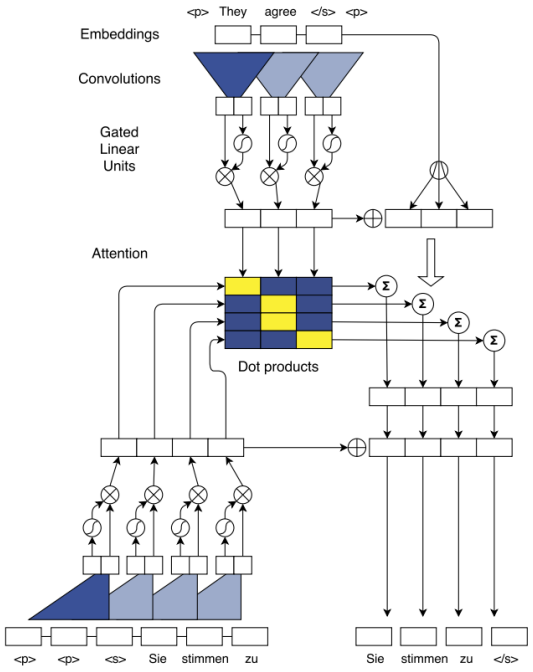

Figure 15: Architecture of the Convolutional Sequence-to-Sequence model introduced by Gehring et al. [2017a]

As noted in the paper of Gehring et al. [2017a], in the process of training, the sequence of words in the source language (English) is encoded and the attention values for all four German target words are computed simultaneously. This is achieved by computing dot products between representations of the decoder context and encoder, and then adding the resulting attention values to the decoder states. These states are then used to predict the target language words, with the use of Gated Linear Units as illustrated by the sigmoid and multiplicative boxes [Gehring et al., 2017a]. In the following sections, detailed information is given about the experimental setup and the hyperparameters of the model.

Framework The fairseq framework [Gehring et al., 2017b], developed by Facebook AI Research, is a toolkit that utilizes the PyTorch library to implement various sequence-to-sequence models for natural language processing tasks. This framework offers pre-trained models and customizable hyperparameter sets to facilitate experimentation. In the context of this study, the fairseq framework was utilized for the implementation of the Convolutional Sequence-to-Sequence model following the methodological steps of Yin et al. [2019].

Experimental Setup For the experimental setup, we followed the study of Yin et al. [2019]. First, we divided each dataset into three parts, with 80% of the data being used for training, 10% for validation, and the remaining 10% for testing. Next, we trained the convolutional sequence-to-sequence model by utilizing 15 layers of convolutional block in both the encoder and decoder. The first 9 layers of the convolutional block had 512 units and convolutional kernel width of 3, followed by 4 layers with 1,024 units and a kernel width of 3, and the final 2 layers with 2,048 units and a kernel width of 1. Additionally, the embedding size was set to 768 for the encoder and the input of the decoder, and 512 for the output of the decoder. The training process employed a fixed learning rate of 0.5 and a dropout rate of 0.2. The maximum number of training iterations is set to 250 epochs, with a batch size of 4,000 tokens. The loss function utilized is label-smoothed cross entropy with a label smoothing rate of 0.1 [Yin et al., 2019].

4 Results

In this section, the results of the project will be presented. First, the evaluation of our model on the LC-QUAD and the Monument dataset is presented in 4.1, while the interface of our web application is in 4.2.

4.1 Evaluation

BLEU is a widely used metric for evaluating neural machine translation systems. It is well-regarded for its strong correlation with human evaluations and its low cost to run [Yin et al., 2019]. It is the most commonly used metric in the field, making it easy to compare results with other models and experiments [Yin et al., 2019]. For those reasons, we decided to use this metric for the evaluation of our model. The results that we acquired after training our model and evaluating it give 60.3 BLEU score for validation, a 58.53 for testing on the LC-QUAD and 97.59 BLEU score for validation, and 97.33 for testing on the Monument dataset. In the table below the BLEU scores for each set are presented along with the number of Epochs.

Dataset	*Valid BLEU*	*Test BLEU*	Epochs
LC-QUAD	60.3	58.5	71
Monument	97.6	97.3	220

Table 1: BLEU scores for validation and test subsets

4.2 Web-Application

The main deliverable of our project, the web-application, uses a simple architecture consisting of a dynamic front-end layer that provides a user interface where the user can ask a question in a natural language (English, in our case). That query is then sent to the back-end layer and fed to the text-to-SPARQL model. The output of the model is a raw (encoded) query and needs to be decoded in SPARQL (we show both to the user as the main objective). Then it is sent to the DBpedia endpoint via API in order to retrieve the results. All the outputs are presented to the user in the interface. In Figure 16 we can see an example of an NL question and output from the TriviaNLQ web application.

(Note: currently the kernel model is optimized for Monument dataset querying, so the questions are supposed to be related to that domain.)

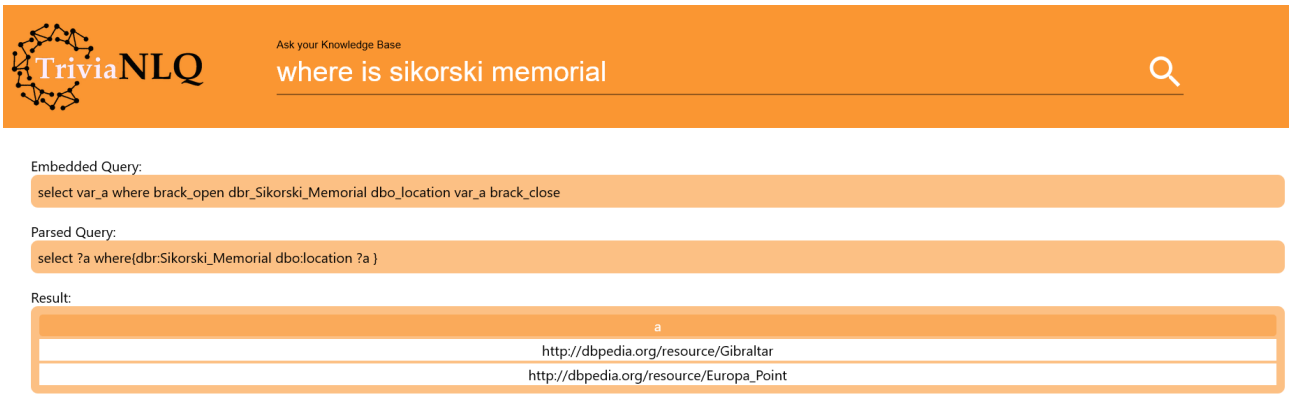

Figure 16: Example of an NL question and output from TriviaNLQ web application

5 Discussion

This paper introduced TriviaNLQ, a web application that utilizes text-to-SPARQL modeling to query the DBpedia knowledge graph and answer trivia questions. Before the implementation of our final model, we first had to test a variety of models, in order to find one that could prove efficient enough to be the basis of our application. This process proved to be of particular importance for gaining an in-depth understanding of the difficulties behind text-to-SPARQL generation, yet extremely time-consuming for the purposes of building an application that performs with high accuracy.

After selecting the model on which we would base our web application, we tested 2 datasets: LC-QUAD and Monument. The LC-QUAD dataset, while high in complexity and compatible with respect to its domain, proved to be not a suitable choice due to its small size in comparison to other datasets commonly used for natural language translation models. The model was unable to learn the proper parameters to fit both the training and validation sets, possibly due to a lack of sufficient training samples to generalize the model to unseen data. For this reason, we tested the second dataset, Monument, which comprised a bigger collection of question-query pairs. Our results suggested a high BLEU score for validation and testing: 97.6, and 97.3 respectively. The model performed significantly better on this dataset and thus we built our web application using it instead of LC-QUAD. Yet, when testing the queries on data on our web application we noticed that many named entities are unknown and thus the result gained from DBPedia is not always accurate. This leads us to conclude that the model is overfitting our dataset and does not perform well on new data. Additionally due to its specification to the domain of monuments, the type of questions we can ask is very specific. The solution to these problems would be to use a larger dataset. DBpedia Neural Question Answering (DBNQA) being one of the largest DBpedia-targeting datasets, could prove to be the most efficient dataset on which we could base our model, yet due to our lacking of training resources, and given the time restrictions of this project it was not possible to make use of this dataset.

6 Limitations and Suggestions for Future Work

Despite the methodological strengths of this study and the high BLEU scores that we got, the results reported herein should be considered in light of some limitations. First, the vocabulary size of our model was small, resulting in low performance in the task of recognizing named entities. This applied to both datasets that we tested. Specifically, due to the limited size of the LC-QUAD dataset, many entities appearing on the NL questions were not recognized, and thus were interpreted by the model as unknown (UNK). This is a common limitation of NMT models being trained on a small dataset. This also applies to the Monument dataset, which we chose as our main dataset due to the fact that it included a bigger size of question-query pairs. However, it also proved to be insufficient. A possible solution to this issue would be to train the model on the DBpedia Neural Question Answering (DBNQA) dataset. In the future steps of this projects we are aiming to train the model on this dataset which will result in better performance of our system on new data.

Another possible limitation of this study is that the queries of our web application pass through a different version of DBpedia from that used in the original trained model of Yin et al. [2019]. This could also explain why our queries are not always giving a proper result from DBpedia. However, it should be noted that in most cases, the syntax of

the SPARQL query is correct, suggesting that the application can be used as a way of transforming NL questions to SPARQL queries despite the final output acquired from DBpedia.

Overall, it is important to note that finding a model that generates SPARQL queries from NL proved to be a challenging task. Despite having tested multiple available models, the results of our search were most of the time not satisfactory. As a result, the process of searching for an efficient model for the purposes of our web application turned out to be time-consuming and particularly difficult. This left us with less time for the implementation and the continuous improvement of our application. Despite the fact that the current application is far from being perfect we still hope that this paper presents some value in comparing different models and approaches for the text-to-SPARQL task.

References

Shiqi Liang, Kurt Stockinger, Tarcisio Mendes de Farias, Maria Anisimova, and Manuel Gil. Querying knowledge graphs in natural language. *Journal of big data*, 8(1):1–23, 2021.

Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural machine translating from natural language to SPARQL. *CoRR*, abs/1906.09302, 2019. URL `http://arxiv.org/abs/1906.09302`.

Merrill Cook. Knowledge base, Apr 2020. URL `https://blog.diffbot.com/knowledge-graph-glossary/knowledge-base/#:~:text=All%20knowledge%20graphs%20are%20knowledge,around%20the%20relationships%20between%20entities.`

Ajitesh Kumar. Knowledge graph concepts & machine learning: Examples, Aug 2022. URL `https://vitalflux.com/knowledge-graph-concepts-machine-learning-examples/#:~:text=An%20example%20of%20a%20knowledge%20graph%20from%20healthcare%20would%20be,disease%20can%20cause%20another%20disease).`

Ontotext Fundamentals. What is a knowledge graph?, Apr 2022. URL `https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/`.

Dieter Fensel. Ontologies. In *Ontologies*, pages 11–18. Springer, 2001.

Renzo Angles and Claudio Gutierrez. The expressive power of sparql. In *International Semantic Web Conference*, pages 114–129. Springer, 2008.

Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks. The semantic web: The roles of xml and rdf. *IEEE Internet computing*, 4(5):63–73, 2000.

Marcin Wylot, Manfred Hauswirth, Philippe Cudré-Mauroux, and Sherif Sakr. Rdf data storage and query processing schemes: A survey. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.

Lee Feigenbaum, Jun 2009. URL `https://www.w3.org/2009/Talks/0615-qbe/`.

Zhixing Tan, Shuo Wang, Zonghan Yang, Gang Chen, Xuancheng Huang, Maosong Sun, and Yang Liu. Neural machine translation: A review of methods, resources, and tools. *AI Open*, 1:5–21, 2020.

Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Jen-Tzung Lai. Statistical methods for machine translation. In *Proceedings of the workshop on speech and natural language*, pages 176–187. Association for Computational Linguistics, 1990.

Philipp Koehn. Statistical phrase-based translation. In *Proceedings of the 2003 conference of the North American chapter of the association for computational linguistics on Human language technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

Kyunghyun Cho and Bart van Merriënboer. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Hieu Hoang, Alexandra Birch, and Mariano Federico. Phrase-based & neural unsupervised machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2858–2868, 2016.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017a.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

Shiqi Shen, Yu Zhao, Zhiyuan Liu, Maosong Sun, et al. Neural headline generation with sentence-wise optimization. *arXiv preprint arXiv:1604.01904*, 2016.

Chen Li, Zhen Zhang, Wee Sun Lee, and Gim Hee Lee. Convolutional sequence to sequence model for human dynamics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5226–5234, 2018.

Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. Neural machine translation (seq2seq) tutorial. 2017.

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.

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.

Fabiano Ferreira Luz and Marcelo Finger. Semantic parsing natural language into sparql: improving target language representation with neural attention. *arXiv preprint arXiv:1803.04329*, 2018.

Machinalis. Machinalis/quepy: A python framework to transform natural language questions to queries in a database query language., 2018. URL `https://github.com/machinalis/quepy`.

Benjamin A Spiegel, Vincent Cheong, James E Kaplan, and Anthony Sanchez. Mk-squit: Synthesizing questions using iterative template-filling. *arXiv preprint arXiv:2011.02566*, 2020.

Yazdipour. Yazdipour/text-to-sparql: End-to-end model - finetuned t5 for text-to-sparql task, 2021. URL `https://github.com/yazdipour/text-to-sparql`.

Irina Saparina and Anton Osokin. Sparqling database queries from intermediate question decompositions. *arXiv preprint arXiv:2109.06162*, 2021.

Rose Hirigoyen, Amal Zouaq, and Samuel Reyd. A copy mechanism for handling knowledge base elements in sparql neural machine translation. *arXiv preprint arXiv:2211.10271*, 2022.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and N Yann. Dauphin: Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252, 2017b.