UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# QUESTION ANSWERING OVER WIKIDATA USING ENTITY LINKING AND NEURAL SEMANTIC PARSING

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

DANIEL ALEJANDRO DIOMEDI PINTO

PROFESOR GUÍA:
AIDAN HOGAN

MIEMBROS DE LA COMISIÓN:
ANDRÉS ABELIUK KIMELMAN
FEDERICO OLMEDO BERÓN
HANS LÖBEL DÍAZ

Este trabajo ha sido parcialmente financiado por el
Instituto Milenio Fundamento de los Datos.

SANTIAGO DE CHILE
2021

# Resumen

El objetivo de *Question Answering* sobre *Knowledge Graphs* (KGQA) es encontrar respuestas para preguntas en lenguaje natural sobre un *Knowledge Graph*. Recientes enfoques de KGQA basados en *Neural Semantic Parsing* adoptan un enfoque de *Neural Machine Translation* (NMT), en el que la pregunta en lenguaje natural se traduce a un lenguaje de consulta estructurado. En este contexto, queremos generar una consulta SPARQL que obtenga las respuestas esperadas al ejecutarse en el *endpoint* del respectivo *Knowledge Graph*.

Sin embargo, el enfoque basado en NMT adolece del problema de falta de vocabulario, en el que los términos de una pregunta pueden no haberse visto durante el entrenamiento, lo que dificulta su traducción. Este fenómeno es particularmente problemático para las millones de entidades que describen los grandes *Knowledge Graphs*. En este trabajo proponemos en cambio un enfoque para KGQA que delega el procesamiento de entidades a sistemas de *Entity Linking* (EL). Por lo tanto, en lugar de generar la consulta SPARQL completa, el modelo de NMT se utiliza para crear un *Query Template* con *placeholders* que se llenan con entidades identificadas en la etapa de EL. Se proponen sistemas EL tipo *ensemble* que combinan resultados de varios sistemas EL individuales del estado del arte. Se propone un enfoque de *Slot Filling* para decidir qué entidad ocupa qué *placeholder*, el cual combina el uso de un modelo de *Sequence Labeling* con un algoritmo de llenado propuesto.

Evaluamos nuestro enfoque en el contexto de Wikidata para preguntas en inglés. Los experimentos evalúan el rendimiento del sistema de *Question Answering* de principio a fin, así como cada etapa de la generación de consultas SPARQL. Los resultados muestran que nuestro enfoque supera al enfoque de NMT puro: aunque sigue existiendo una fuerte dependencia en haber visto *Query Templates* similares durante el entrenamiento, los errores relacionados con las entidades se reducen en gran medida.

La principal conclusión es que la combinación de *Entity Linking* y *Neural Semantic Parsing* muestra una mejora prometedora en el rendimiento de la tarea KGQA en el contexto de Wikidata. El trabajo futuro incluye experimentar con otros modelos de NMT como también trabajar en la construcción de conjuntos de datos de entrenamiento de mejor calidad, agregar nuevos sistemas EL para impulsar los sistemas tipo *ensemble* y probar nuevas heurísticas para el proceso de *Slot Filling*.

# Abstract

The goal of *Question Answering over Knowledge Graphs* (KGQA) is to find answers for natural language questions over a *Knowledge Graph.* Recent Neural Semantic Parsing-based KGQA approaches adopt a *Neural Machine Translation* (NMT) approach, where the natural language question is translated into a structured query language. In this context, we want to generate a SPARQL query that should retrieve the expected answers by being executed over any Knowledge Graph's endpoint service.

However, the approach based on NMT suffers from the out-of-vocabulary problem, where terms in a question may not have been seen during training, impeding their translation. This issue is particularly problematic for the millions of entities that large Knowledge Graphs describe. We rather propose a KGQA approach that delegates the processing of entities to *Entity Linking* (EL) systems. Thereby, instead of generating the entire SPARQL query, the NMT is used to create a Query Template with placeholders that are filled by entities identified in an EL phase. We propose ensemble EL systems that combine output from several individual EL systems from the state of the art. A *Slot filling* approach is proposed to decide which entity fills which placeholder, combining a *Sequence Labeling* model with a proposed filling algorithm.

We evaluate our approach in the context of Wikidata for English questions. Experiments evaluate the performance of the end-to-end QA system as well as each stage of the SPARQL query generation. The results show that our approach outperforms the pure NMT approach: while there remains a strong dependence on having seen similar Query Templates during training, errors relating to entities are greatly reduced.

The main conclusion is that the combination of Entity Linking and Neural Semantic Parsing shows a promising improvement in the performance of the KGQA task in the context of Wikidata. Future work includes experimenting with other NMT models while also working on building better quality training datasets, adding new EL systems for boosting ensemble systems, and trying new heuristics for the Slot Filling process.

*Para* Delia Pinto *y* Alexis Diomedi*, por ser*
*un pilar fundamental en este hito cumplido.*

# Agradecimientos

Con este trabajo culmina una etapa que duró casi 7 años de mi vida, y por supuesto hubo harta gente que me brindó su apoyo que fue indispensable y muy valioso. Esta página es para todos y todas ustedes que estuvieron ahí.

Partir con agradecer a mi familia, que siempre me brindó todo lo necesario para que pudiese dar lo mejor de mí durante mis estudios. Agradezco su comprensión en momentos en que la universidad no dejaba otra cosa que ser la prioridad. Fueron, han sido, y serán la base de todo lo que he logrado durante mi estancia en la universidad y los logros que se vienen.

No todo fue seriedad y estudio por supuesto. Los momentos de distensión también fueron muy valiosos y donde guardo los mejores recuerdos es con la gente del Pasillo. Muchas gracias a todos y todas los que se pasaron aunque sea una vez por el pasillo por cada almuerzo compartido y por cada conversación llena de risas, complicidad e irreverencia.

También mencionar a toda la gente maravillosa con la que pude compartir en la universidad, en especial la gente del DCC. Aunque no solía frecuentar muchos espacios de estudios, siempre fue un gusto conversar con cada persona que me crucé por esos lares. Mención especial a los manDCCos por no solo las tarde de estudio, si no también todas esas noches de juegos hasta la madrugada (o no tanto dado que "dormir temprano" es mi segundo nombre).

Una tremenda mención honrosa a Sandra y Angélica por aguantar tantos semestres de mi haciendo preguntas y buscando consejo. Siempre tuvieron la mejor de las disposiciones y no encuentro la manera de agradecer por su valioso apoyo durante todo este tiempo.

Esta tesis no hubiera salido sin la indispensable ayuda de mi profesor guía Aidan. Aunque habían semanas en las que sentía que no había avance, su invaluable guía siempre me entregaba una perspectiva alentadora a los desafíos que iban saliendo. Agradezco a Juan Pablo Silva por escuchar mis delirios con la tesis, a Henry Rosales por su asesoría en sistemas de Entity Linking, a Alberto Moya por su ayuda en montar Wikidata en Virtuoso, a Felipe Bravo y Jorge Peréz por el espacio para compartir mi trabajo y por el feedback entregado.

Y gracias a ti, Javiera Francisca, por ser y estar.

# Tabla de Contenido

# Índice de Tablas

# Índice de Ilustraciones

# Chapter 1

# Introduction

## 1.1. Motivation

The volume of knowledge found on the web is growing considerably, so the interest of many communities is in profiting from that knowledge. Many questions are being asked to search engines like Google, which serves roughly 4.2 million searches done by users every minute[1]. Since most of the data found on the web does not have a standard structure, search engines do not tend to reply to the question directly but just to retrieve the documents that might contain the answer. Though many questions can be answered by doing so, many other more complex questions require a higher level of reasoning that is difficult to achieve by consulting only unstructured data. Thus, there is still a challenging problem with making data more accessible, even knowing that its volume is increasing exponentially.

To give semantic meaning to all this information available on the Web in a manner in which both humans and machines can understand, a common framework is required. Thus, the Semantic Web [19] was proposed as an extension of the World Wide Web built on standards set by the World Wide Web Consortium (W3C). The primary purpose of this initiative is to support a *"Web of Data"* where data can be searched like in databases, but at the scope of the Web. The compilation of Semantic Web techniques and tools provides a framework where applications can query that data, draw inferences using vocabulary, etc. Thus, the ultimate goal is to extend the variety of tasks that computational systems can support, while developing trusted interactions over the network.

The Semantic Web establishes a standard method to describe data using the Resource Description Framework (`RDF`) [157]. This data model describes resources using statements of the form subject-predicate-object, also called triples, and can be represented as a directed edge-labelled graph. A collection of `RDF` statements is known as a Knowledge Graph

[1] https://www.internetlivestats.com/

(KG) [84]. Altogether, these KGs when linked together on the Web give shape to what is called the Linked Data Cloud [17]: a large amount of interlinked RDF datasets that comprise more than 30 billion RDF triples. Among the most popular KGs, Wikidata [183] and DBpedia [107] are huge and become more useful and accessible each day for research fields and applications [120, 128].

Wikidata [183] is a free open Knowledge Graph that can be read and edited by both humans and machines. Since the Wikimedia Foundation first launched Wikidata in October 2012, it has grown vastly. It has served as a reference resource for many of Wikimedia's sister projects, like Wikipedia, the largest virtual encyclopedia on the Web. Wikidata is a valuable and comprehensive source of knowledge. It is supported mainly by its community and is designed in a way that people from all over the world can contribute. Many applications have used Wikidata as an information provider such as Apple's Siri; it has also been used for research activities in life sciences and social science, and it even is used by Google to empower its search engine [120].

Thereafter, for querying this vast amount of data available on the web, a query language is needed. SPARQL [80] is a query language able to retrieve and manipulate data stored in RDF format. The main advantages of SPARQL are that it allows for writing queries that follow RDF specifications and provides a specific graph transversal syntax for querying arbitrary-length paths in graphs.

While all of this knowledge available in the public domain drives growing interest in doing research regarding the Semantic Web, there is also a need to have a basic understanding of how data is structured (RDF) and how to access this data (SPARQL). These requirements represent a barrier-to-entry for non-expert users. Hence these barriers lead to the broad and complex challenge of developing intuitive and easy-to-use interfaces for end-users.

Many solutions have emerged to approach this issue, among which natural language interfaces such as Question Answering Systems (QASs) [26, 177, 31] have been receiving much attention. These systems aim to answer questions posed by humans in natural language, extracting the answer from one or more sources. There are QASs able to retrieve answers from an unstructured collection of natural language [26]. However, we are interested in systems able to construct their responses by querying structured data, like relational databases or knowledge graphs from the Linked Data Cloud.

More specifically, the task of answering natural language questions using Knowledge Graphs is known as Question Answering over Knowledge Graphs (KGQA) [31] or Question Answering over Linked Data (QALD) [177, 112]. Unger et al. [177] define the QALD task as follows: *"translate the users' information need into a form such that they can be evaluated using standard Semantic Web query processing and inference techniques"*. Their work also describes the types of questions these systems aim to answer, which often focus on definition questions (*"Who was Tom Jobim?"*) or factoid questions. These last ones that can be divided into predicative questions (*"Who was the first man in space?"*), list questions (*"Give me all cities in Germany"*) or boolean questions (*"Was Margaret Thatcher a chemist?"*).

Several Question Answering systems have been developed to address some of the main challenges in Question Answering, commonly using pattern matching [50, 111], grammar-based techniques [40, 123], or graph exploration [191, 205] approaches. Although these systems have shown good results when dealing with a considerable amount of questions, their performance decreases [90] with questions involving more complex graph patterns or with vocabulary mismatches caused by the user typing different terms to the ones contained in the Knowledge Graph from which the information is being retrieved (this issue is also known as the lexical gap [77]). One example is the question *"Which US player is the highest scorer in World Cups?"* which could require complex operations like aggregation and sorting (count goals, sort and retrieve the maximum scorer) and might have vocabulary mismatches (it is not made explicit that we refer to FIFA World Cups, or the US might not be registered as an abbreviation of United States).

Aside from works that have tried to mitigate these problems [77, 66], some approaches that rely on Semantic Parsing have shown positive results due to recent advances in Deep Learning applied to Natural Language Processing [8]. Semantic Parsing is the process of mapping a natural language sentence into a formal representation of its meaning [8]. Some applications include code generation [149, 196], automated reasoning [99] or query construction [55]. In particular, Andreas et al. [8] discussed how Semantic Parsing could benefit from using Machine Translation methods, whereby natural language is "translated" into a structured representation. Following their work, some Neural Machine Translation (NMT) approaches have brought about a growing interest in applying deep neural networks to Semantic Parsing problems [30, 44, 204]. In NMT, pairs of sequences are given as input to a Deep Neural Network model, which is expected to learn the translation model. A natural idea is then to try to apply the NMT approach for translating Natural Language (NL) to `SPARQL`, and some works have begun to explore such techniques [116, 162, 163].

There are multiple challenges relating to converting a NL question to its `SPARQL` query representation (`NL-to-SPARQL`); some of these challenges are directly inherited from the original NL-to-NL translation problem, while others are distinct. First, there isn't a one-to-one mapping for every NL question to a `SPARQL` query. On one hand, there are multiple ways to express a question in NL. For example, the question *"How far away is the Earth from the Sun?"* can also be rephrased as *"What is the distance between the Sun and Earth?"*. On the other hand, questions can be translated to `SPARQL` in different ways. For example, a question *"What is the largest country in Africa?"* might be translated to a query based on population or area, where one such translation must be chosen, and where both give different answers. Moreover, one `SPARQL` query has potentially equivalent queries that will return the same results (over any data). In the question about US players, for example, we can first retrieve US players and then count their scores, or vice versa; thus there is a need to establish some common conventions when designing `NL-to-SPARQL` systems.

Another issue is the lack of corpora for training NL-to-SPARQL models when compared to the enormous amount of documents in different languages that can be found on the Web for training NL-to-NL models (e.g. news, blogs, articles, academic documents, etc.). Generating data for NL-to-SPARQL is not an easy task considering the need for a basic SPARQL understanding to build such datasets, although there is some work regarding automating parts of the process of generating NL-to-SPARQL pairs [81, 174]. Furthermore, the queries required to answer a question change from dataset to dataset, where the SPARQL queries needed for DBpedia are not the same as those for Wikidata.

One of the most recent works in regards to using NMT for SPARQL is that of the *Neural SPARQL Machine* (NSpM) [162], which considers SPARQL as a foreign language. The main idea is to train an end-to-end learning model to translate any NL expression into a sequence of tokens in the SPARQL grammar that expresses a query equivalent to the question over a given dataset. Question Answering systems based on neural networks usually aim to generate the whole SPARQL query in an attempt to perform the entire process of identifying the relevant entities along with deducing the KG properties, triples, and operators that would retrieve the expected answer.

An analysis of the performance of many NMT models on translating NL to SPARQL has been performed by Yin et al. [197], where eight deep neural network models were tested over different datasets based on questions over DBpedia. One relevant dataset is the *Large Complex Question Answering Dataset* (LC-QuAD 1) [174], consisting of 5,000 complex questions based on 38 hand-made templates. Another dataset is the *DBpedia Neural Question Answering* dataset (DBNQA) [81] that includes almost 900,000 questions based on templates extracted from questions of the LC-QuAD 1 dataset and the 7th version of the *Question Answering over Linked Data* dataset (QALD-7) [178].

Though the performance of these models shows promising potential for constructing meaningful and useful SPARQL queries, they present some key limitations relating to the data used to train the models. First, despite the fact that many NMT models report positive results when evaluated over simple and large datasets like the DBNQA dataset [197], which contains questions with little variation in syntax and phrasing, such regular data do not give an accurate understanding of the real performance of these systems. In fact, the performance of such models drops dramatically when evaluated over more complex questions like the ones included in the LC-QuAD 1 dataset, which might not contain enough questions to learn accurately [197]. Second, the current models are vocabulary-dependent, which means there is no capability for recognizing new entities or properties that were not used in the training data [197]. These issues lead to the constant need to train the model with new, manually created pairs of NL questions and SPARQL queries.

The first issue can be addressed by building a more comprehensive dataset that has enough cases for an NMT to learn properly while maintaining an abstraction level that allows us to respond accurately to complex and diverse questions. Regardless, creating new datasets does not necessarily help with the vocabulary dependency issue, unless we have examples using every entity and property in the Knowledge Graph, which seems infeasible to generate in the short-to-medium term. Therefore, an important goal is to maximize the available training examples, where there is a need to find an alternative approach to complement the parsing power of NMTs with a system that helps to address vocabulary dependency by extracting the information NMTs cannot generalize.

For example, NMTs cannot be expected to extract entities from a question and translate them to their identifiers in the KG. Developing labelled examples for each entity does not seem feasible, as mentioned before. Plenty of solutions have addressed the entity extraction task over Linked Data. In particular, the Information Extraction area, which involves the automatic extraction of implicit information from unstructured or semi-structured sources, intersects in many ways with the Semantic Web [124]. Some examples are systems that perform Named Entity Recognition [104], Sequence Labeling [117, 5], or Entity Linking [128, 198, 52, 42] while leveraging Semantic Web resources and/or techniques. In particular, Entity Linking (EL) systems aim to perform the entire process of identifying entity names in a text, mapping names to KG resources, and disambiguating them depending on the context of a given corpus [189]. Considering again the question *"Which US player is the highest scorer in World Cups?"*, an EL system could effectively identify the resources associated with the country US or the FIFA World Cup. Many EL systems [128, 198, 52, 42] have achieved positive results when linking entities over text and these systems tend to generalize well over any new corpus.

Furthermore, these Information Extraction tools can be complemented with intermediate representation of structured queries. An example can be found in a proposed Text-to-SQL system [55], where given a question in NL, an intermediate representation of an `SQL` query is generated, consisting of a `SQL` template with slots to be filled later with relevant words identified in the question using Named Entity Recognition tools [134].

We see an opportunity to improve state-of-the-art performance for QASs in the context of `RDF`/`SPARQL` by exploring the idea of combining the parsing capacity of NMT to get an intermediate representation of a `SPARQL` query, with the entity extraction and disambiguation power of Entity Linking systems to identify the relevant entities in questions, decreasing the dependency of current QA systems on training examples that cover the full vocabulary of a Knowledge Graph. Additionally, there are many challenges to address – as we have previously mentioned – like how to deal with different representations of the same question (e.g. paraphrased questions), what canonical representation of `SPARQL` queries we should adopt, how to evaluate that a QAS is fulfilling its purposes, among others.

## 1.2. Hypothesis

In this work, we propose the following hypothesis: *a combination of Information Extraction with Semantic Parsing can develop a Question-Answering system that outperforms a* *system that relies only on Semantic Parsing in the Question Answering over Knowledge Graphs task.*

In order to measure performance, this work will use metrics based on two perspectives: one focuses on the final answers that are derived from Question Answering over Knowledge Graphs (KGQA) benchmarks (i.e., a Question Answering-based evaluation), and the second focuses on how close is the generated `SPARQL` query compared with the expected query (i.e., a Machine Translation-based evaluation).

The scope of this work will be limited to answering questions in English, but similar techniques should be applicable in any other language assuming the availability of similar datasets for that language. In the same way, this hypothesis will be explored in the context of questions over Wikidata, so the results might differ for other Knowledge Graphs. Nevertheless, the selected approach should be generalizable to other domains.

## 1.3. Objectives

### General Objective

We aim to improve upon state-of-art Question Answering systems based on Neural Semantic Parsing models by reducing vocabulary dependency on the data used in the learning process of such models.

### Specific Objectives

The specific objective is to build a Question-Answering system over Wikidata in English, that relies on Entity Linking and Neural Machine Translation systems, with an intermediate system that combines both tools. Our initial claim is that such a system can improve upon the state-of-the-art Neural Machine Translation approaches found in the literature.

## 1.4. Methodology

Accomplishing the proposed objectives involves the following tasks:

- Survey previous work regarding Neural Machine Translation, Entity Linking, and Question Answering approaches that rely on Neural Machine Translation.

- Define a benchmark that should include KGQA datasets for training, validation and testing along with metrics to compare all involved systems.

- Define a baseline system for Question Answering based on Neural Machine Translation.

- Define a pipeline process to convert a natural language question into a SPARQL query by combining Entity Linking techniques with Neural Machine Translation.

- Implement a Question-Answering system over Wikidata in English based on the designed pipeline.

- Validate the proposed system by comparing it with baseline approaches over the proposed benchmark.

## 1.5. Contributions

We present the three main contributions anticipated for this work.

### Benchmark on Question Answering over Wikidata

After a bibliographic revision, we will define a benchmark as a combination of three components: a baseline system, a set of metrics to compare with the baseline, and one or more datasets with which to conduct experiments.

The baseline consists of one of the Neural Machine Translation systems described by Yin et al. [197]. From the eight models that were compared in this work, the ConvS2S model [61] significantly outperforms all the other models . Following these results, a baseline QAS is implemented using the Fairseq library [137], which includes a ConvS2S implementation over Pytorch[2] ready to use for training and translation. The model is trained using the same settings described by Yin et al.

The primary dataset used is the LC-QuAD 2 dataset, which contains around $30,000$ questions over Wikidata [47]. A quality check is performed over this dataset, where cases that contain either invalid questions or invalid SPARQL queries are filtered. The DBNQA dataset [81] is considered as well, where a mapping process is applied to obtain queries over Wikidata. The queries that cannot be mapped are ignored. The Question Answering over Linked Data (QALD) [112] dataset is also used as part of this benchmark. In particular, the 150 questions included in QALD-7 [178] that can be answered over Wikidata are considered. Besides these datasets, we build a new dataset of 100 questions over Wikidata. Only LC-QuAD 2 and the mapped version of DBNQA are used for training and validation. The other datasets are used only for testing purposes. All datasets are arranged to follow a common format, which will permit an easy evaluation of every subtask performed for the proposed Question Answering system (Entity Linking, Query Template Generation, Slot Filling) along with the main task (Question Answering over Knowledge Graphs).

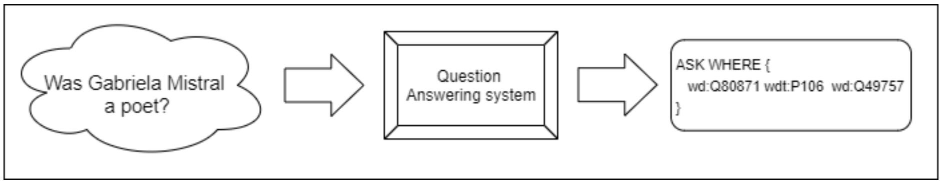
The metrics used for comparing systems are based on the ones used for comparing Neural Machine Translation systems and the ones found on the QALD benchmark. The first set of

[2] https://pytorch.org/

metrics includes the BLEU score, Perplexity, and Accuracy by comparing an exact match on the `SPARQL` query. On the other hand, the QALD benchmark uses Precision, Recall, and F1-score over the answers obtained when executing the output `SPARQL` query. Additionally, we propose a fine-grained analysis over each case where predicted queries are evaluated with respect to the following components: correct entities, correct slots, and correct query templates.

## Question Answering system over Wikidata

The main contribution of this work is a Question Answering system that translates natural language questions in English to `SPARQL` queries executable on Wikidata endpoints. See an example of an expected `SPARQL` query in Figure 1.1. The implementation of this system is divided into the construction of three modules: a Query Template generator, an Entity Linking system, and an intermediate Slot Filling system.



```
ASK WHERE {
  wd:Q80871 wdt:P106  wd:Q49757
}
```

Figure 1.1: Expected `SPARQL` query example from a KGQA system.

The Query Template generator produces incomplete `SPARQL` queries (which we call Query Templates) that contain *"placeholders"* in the position where entities are supposed to be (e.g. placeholders `<sbj_1>` and `<obj_1>` instead of entities `Q80871` and `Q49757` of the query shown in Figure 1.1). This module is built using the same model used to implement the baseline QAS. However, the training data is adapted to generate Query Templates instead of the complete query. This is achieved by removing the entities from the output `SPARQL` queries included in the selected datasets such that the entities can rather be found by Entity Linking.

The role of the Entity Linking module is to recognize the relevant entities contained in each question (e.g. identify that concepts *"Gabriela Mistral"* and *"poet"* corresponds to the entities `Q80871` and `Q49757` in Figure 1.1) that are used to fill the Query Template. We implement various entity retrieval systems using one or more of the existing Entity Linking systems that have APIs available. The first variant is to use each EL system individually, which includes DBpedia Spotlight [128], AIDA [198], TAGME [52], and OpenTapioca [42]. All of these systems, except for OpenTapioca, only work for DBpedia; therefore an extra mapping layer is implemented to map DBpedia entities to Wikidata entities. Two ensemble EL approaches are then proposed: one that prioritizes systems with higher Precision and another that implements a voting mechanism. We keep the variant that performs best according to the experiments that are explained in the *Experimental results* subsection.

Additionally, a Slot Filling module is built by combining a Sequence Tagger model of the Flair library [4] and a filling algorithm proposed in this work. Training the Sequence Tagger model requires building training data based on the selected datasets. Intuitively speaking, a Query Template may have multiple slots and multiple entities, where the Slot Filling module decides which entity should fill which slot (e.g. to infer that the concept *"Gabriela Mistral"* corresponds to the placeholder `<sbj_1>`, therefore the entity `Q80871` should replace the occurrences of `<sbj_1>` in the Query Template).

### Experimental results

We conduct several experiments for validating each implemented module (Entity Linking, Slot Filling, and Query Template Generation) along with experiments over the defined benchmark for the end-to-end Question Answering process.

The Entity Linking systems are compared using Precision, Recall, and F1-score on the entities for each case on the dataset used for training. Testing is conducted over `QALD-7` and our proposed dataset. The slot filling system is validated using Precision, Recall, and F1-score over the identified BIO labels (a common tagging format for sequence labeling tasks) over `LC-QuAD 2` and the mapped `DBNQA` dataset. The query generator system is validated using BLEU score, Perplexity, and Accuracy over `LC-QuAD 2` and the mapped `DBNQA` dataset. When training the query generator system, many split methods are tested according to the methodology proposed by Finegan-Dollak et al. [55] for Text-to-SQL systems. The end-to-end Question Answering system is tested over all the datasets using the metrics described in the *Benchmark on Question Answering over Wikidata* subsection.

## 1.6. Work Structure

This worked is divided into the following chapters:

1. In Chapter 2, we describe the theoretical framework enclosed on this work. This chapter the Semantic Web, Information Extraction methods and how they relate with Semantic Web technologies, Semantic Parsing applied to translating natural language to `SPARQL`, and the current state and challenges of the Question Answering over Knowledge Graphs task.

2. In Chapter 3, we give an overview of the proposed Question Answering system for this work. This includes a general explanation of the pipeline proposed to generate a `SPARQL` query, and more specific details on how each component is designed.

3. In Chapter 4, we go into details about the experiments we run in this work. We present the research questions we aimed to answer, the baseline we compare our system with, and the metrics used to quantify the performance of each system.

4. In Chapter 5, we present the results derived from running the proposed experiments. Aside from that, we include a brief discussion and analysis of the results.

5. In Chapter 6, we summarize the conclusion of this work, discuss its limitations and the future work regarding Question Answering over Knowledge Graphs.

# Chapter 2

# Theoretical Framework

## 2.1. Semantic Web

### 2.1.1. Web of Data

During the short history of the World Wide Web, we have greatly benefited from how its content has been increasing every day while serving multiple purposes. This vast amount of knowledge has become humanly impossible to traverse, so we rely on machines to process the content of documents automatically. As Hogan mentions, machines require the data to fulfill two primary requirements in order to be able to process them automatically in a meaningful way: to have a machine-readable structure and semantics [91]. Unless a more “formal” notion of structure and semantics is provided, machines can not be used to their full potential.

Various standards have emerged to partially structure the Web’s content, such as XML, CSV, or JSON. However, structured content without some semantics does not permit machines to do much more than split the content up by its delimiters and load its structure. Some other standards define semantic meaning for their structure, where some prominent examples are HTML, RSS or XML Schema (XSD), but often the semantics are defined in a human-readable way, for example, to describe how certain HTML elements should be rendered in a browser. Though these markup-based specifications provide a set of terms that often serve a singular purpose within the context of a given application, their interpretation tends to differ significantly for the respective consumer applications.

The multiple purposes that standards like HTML can serve are not able to address some of the shortcomings of the current Web. Since content is often created to serve specific functionalities in the context of a given site, much of this content ends up not being directly reusable, high levels of redundancy appear or it cannot be integrated with other sites. In an effort to address these limitations, the Semantic Web was proposed.

The Semantic Web is designed as an extension of the current World Wide Web so as to enable the creation, sharing, and intelligent re-use of machine-readable content on the Web. The inception of the modern notion of the Semantic Web is founded on two major milestones: the original W3C recommendation of the first Resource Description Framework (`RDF`) standard defining the core data model [105], and the introduction of the vision for the Semantic Web outlined by Berners-Lee et al. [19].

The vision of the Semantic Web can be represented through the Semantic Web Stack, first conceived by Berners-Lee et al., as seen in Figure 2.1. The lower levels describe the foundational elements of the Semantic Web which are aligned with the Web itself. First, the Web needs some standard to map from binary-streams and storage to textual information, so it relies on **Characters** from the standard Unicode character-set. Then, **Identifiers** respond to the main purposes of denoting any concept or concrete thing. The natural choice is to use Uniform Resource Identifiers (URI), which is the native standard for identification on the Web, or a recently adopted generalization of URI, Internationalized Resource Identifiers (IRI), which additionally support unescaped Unicode strings. The **Syntax** layer serves the objective of allowing machines to automatically parse content into its elementary components by defining syntaxes with formally defined grammars. For example, one of the most common syntaxes used to encode Semantic Web data is the Terse RDF Triple Language (Turtle) syntax [15], based on the Notation3 (N3) syntax [18].

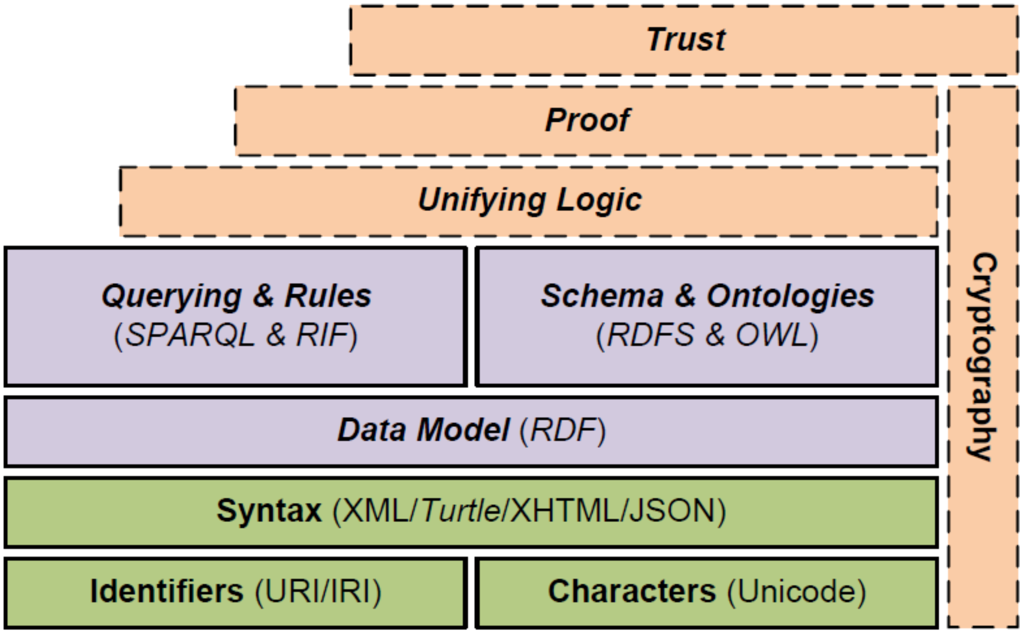

Trust
Proof
Unifying Logic
Querying & Rules (SPARQL & RIF)
Schema & Ontologies (RDFS & OWL)
Data Model (RDF)
Syntax (XML/Turtle/XHTML/JSON)
Identifiers (URI/IRI)
Characters (Unicode)
Cryptography

Figure 2.1: Semantic Web Stack [91]

The next set of layers forms the core of the Semantic Web. The **Data Model** layer provides a canonical representation where machines can exchange machine-readable data in a generic framework. The core data-model for the Semantic Web is the Resource Description Framework (RDF) [121]. In order to bring some meaning to the RDF content, it requires formal languages whose meta-vocabulary complement the RDF data-model by providing well-defined semantics. These languages correspond to the **Schema & Ontologies** layer, where the RDF Schema (RDFS) [105] and Web Ontology Language (OWL) [125, 69] standards are the essential languages integrated as part of the current Semantic Web. Eventually, the content described in RDF needs to be processed by declarative querying and rules languages that serve many purposes, like generating results for user interfaces or inferring novel RDF data. The **Querying & Rules** layer is where the querying and rules standards for the Semantic Web are defined. In this case, the *SPARQL Protocol and RDF Query Language* [147, 51, 80] defines the querying standards and the Rule Interchange Format (RIF) [101] defines the rules standards.

The top and side of the stack in Figure 2.1 are layers yet to be realized. Though many proposals have been made in the research literature, no mature standards or tooling have emerged. The remaining top layers aim to combine the described low-level technologies into a unifying language to execute queries and rules over knowledge represented in RDF (Unifying Logic), provide proofs to validate procedures or information used (Proof), and determine the trustworthiness of information sources (Trust). The Cryptography side layer is centered on cryptographic techniques for verifying and allowing access control mechanisms.

Providing more details on this broad overview of the Semantic Web, we start by focusing on how data is represented and how querying the content is described by any selected source. Then, in the following sections, we go deeper into understanding the Semantic Web data-model, RDF, and its querying standard, SPARQL.

### 2.1.2. Resource Description Framework

The **Resource Description Framework** (RDF) standard [121] provides a data-model on the Semantic Web, which can be serialized using the Turtle syntax [15]. Having this data-model allows for any content framed in RDF to be generically processed and indexed by external systems, whatever its topic or origin. The atomic elements that constitute the RDF data-model are called RDF **Terms**. RDF does not follow the Unique Name Assumption (UNA), so two RDF terms can refer to the same referent. The set of RDF terms are divided into three disjoint subsets: URIs, Literals and Blank Nodes.

As mentioned previously, **Uniform Resource Identifiers** serve as identification for any resource. An example to identify the country Chile in DBpedia [107] is the URI `http://dbpedia.org/resource/Chile`. A shorter version can be used by using the CURIE-style shortcuts [20], where a re-usable prefix can be defined: `@prefix dbr: <http://dbpedia.org/resource/>`. This way, the identifier of Chile can be abbreviated to `dbr:Chile`.

**Literal** values represent lexical values and are divided into two categories. Plain literals are a set of plain strings, such as `‘‘Hello World’’`, and can include an associated language tag, such as `‘‘Hello World’’@en`. Typed literals are literals that include a datatype, such as `‘‘8’’^^xsd:int`. Datatypes are identified by URIs (such as `xsd:int`) and borrow most of the datatypes defined for XML Schema [142]. Datatypes are often used for data validation or mapping.

**Blank Nodes** are used as existential variables that denote the existence of some resource without having to explicitly reference it using a URI or literal. The scope of a blank node is limited to the local `RDF` document where it is defined, so it cannot be referenced elsewhere. In Turtle, blank nodes can be referenced explicitly with an underscore prefix `_:bnode1` or implicitly in a variety of other manners.

`RDF` terms are then combined together to form `RDF` Triples. As its name suggests, a triple is a 3-tuple of `RDF` terms. The three components of a triple are commonly called subject, predicate and object. `RDF` triples can be seen as an atomic representation of a “fact” or “claim”, e.g. *“Santiago is the capital city of Chile”*. Typically each `RDF` triple position fulfills a certain role: the subject is the primary resource that is being described (either a URI or a blank node), the predicate is the relation between the subject and the object (must be a URI), and the object is the value of the relation (any of the mentioned `RDF` terms). For instance, we illustrate the Turtle representation of a set of `RDF` triples from DBpedia in Listing 2.1.

```
# PREFIX DECLARATIONS
@prefix dbr: <http://dbpedia.org/resource/>
@prefix dbo: <http://dbpedia.org/resource/>
@prefix dbp: <http://dbpedia.org/resource/>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

# RDF TRIPLES
dbr:Gabriela_Mistral rdfs:label "Gabriela Mistral"@en .
dbr:Gabriela_Mistral dbp:occupation dbr:Poet .
dbr:Gabriela_Mistral dbo:birthPlace dbr:Vicuña,_Chile .
dbr:Gabriela_Mistral dbo:awards dbr:Nobel_Prize_in_Literature .
dbr:Gabriela_Mistral dbo:awards dbr:National_Prize_for_Literature_(Chile) .
dbr:Vicuña,_Chile rdfs:label "Vicuna, Chile"@en .
dbr:Vicuña,_Chile dbo:populationTotal 25085 .
dbr:Vicuña,_Chile dbo:isPartOf dbr:Elqui_Province .
dbr:Vicuña,_Chile dbo:isPartOf dbr:Coquimbo .
dbr:Vicuña,_Chile dbo:country dbr:Chile .
```

Listing 2.1: Set of `RDF` triples about Gabriela Mistral in DBpedia using Turtle syntax.

Listing 2.1 shows facts about Gabriela Mistral, a famous Chilean poet, and how those facts are structured as a set of `RDF` triples, where each triple is separated by a dot symbol. The use of CURIE-style prefixes helps to simplify the content and make it easier to understand for a human reader [20]. Moreover, Listing 2.2 shows how Turtle allows for abbreviating the content by grouping triples with common subjects (using the '`;`' symbol) or predicates (using the '`,`' symbol).

```
...
# RDF TRIPLES
dbr:Gabriela_Mistral rdfs:label "Gabriela Mistral"@en ;
    dbp:occupation dbr:Poet ;
    dbo:birthPlace dbr:Vicuña,_Chile ;
    dbo:awards dbr:Nobel_Prize_in_Literature, dbr:National_Prize_for_Literature_(
        ↪ Chile) .
dbr:Vicuña,_Chile rdfs:label "Vicuña, Chile"@en ;
    dbo:populationTotal 25085 ;
    dbo:isPartOf dbr:Elqui_Province , dbr:Coquimbo ;
    dbo:country dbr:Chile .
```

Listing 2.2: Set of `RDF` abbreviated triples about Gabriela Mistral in DBpedia.

Given the triple-based structure of `RDF` triples, it is possible to represent entire datasets as an `RDF` graph, also known as a **Knowledge Graph** (KG): a directed labeled graph where subjects and objects are represented by nodes, and predicates are represented by the directed edges that bond two nodes. The `RDF` graph of the example shown above can be drawn as in the diagram of Figure 2.2, where by convention ellipses correspond to URIs or blank nodes, and rectangles symbolize literals.

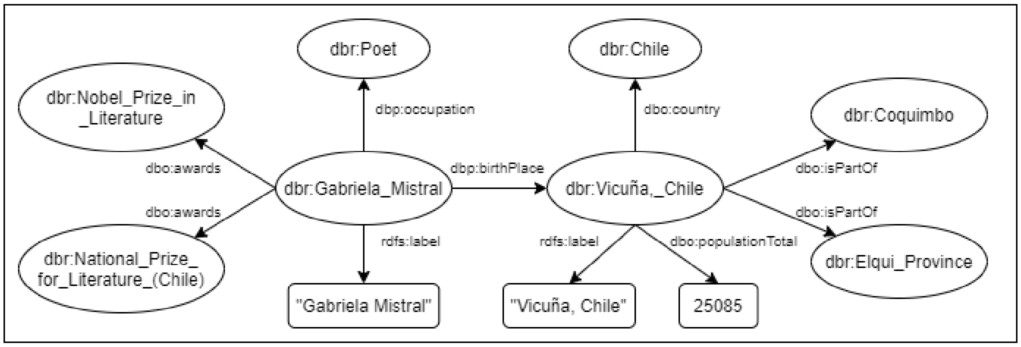


Figure 2.2: `RDF` graph representing facts about Gabriela Mistral from Listing 2.1.

Besides `RDF` triples and `RDF` graphs, `RDF` standards provide a rich set of built-in vocabulary terms under a core `RDF` namespace, which helps to standardize frequently used `RDF` patterns. One popular term is `rdf:type`, which helps to assign resources sharing certain commonalities into **classes**. For example, we can denote `dbr:Vicuna_Chile` as a city by combining it with the predicate `rdf:type` and the object `dbr:City`. Another example is the term `rdfs:label` that comes from RDF Schema [27], which provides other resources for describing relations such as **subproperties** or **subclasses**. Though our work is not focused on the use of the Web Ontology Language (OWL) [69, 125], it is worth mentioning that it also adds a wealth of new vocabulary to describe new relations between resources like equivalences, disjointness, inverse properties, among others.

Ultimately, there are numerous syntaxes for writing RDF apart from Turtle [15], among which we can name RDF/XML [14], N-Triples [71], RDFa [85, 2], and JSON LD [164]. However, no matter what syntax is chosen, every one is represented in the same RDF data model; thus it is possible to convert any RDF content in one syntax to another while keeping the same RDF data.

### 2.1.3. SPARQL Query Language

The **SPARQL** protocol defines how SPARQL queries retrieve results over the RDF datamodel [51]. These standards became W3C Recommendations in 2008 [147] and then were extended in 2013 in the SPARQL 1.1 version [80] superseding the previous W3C Recommendations. Some SPARQL query features and keywords are similar to the ones found in the Structured Query Language (SQL), though SPARQL is designed for interacting with RDF data.

```
# PREFIX DECLARATIONS
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
# DATASET CLAUSE
FROM <http://dbepdia.org/data/Gabriela_Mistral.ttl>
# RESULT CLAUSE
SELECT ?people ?award
# QUERY CLAUSE
WHERE {
    ?people dbo:birthPlace dbr:Vicuña,_Chile ;
        dbo:award ?award .
}
# SOLUTION MODIFIERS
LIMIT 2
```

Listing 2.3: SPARQL query for getting awards won by people who were born in Vicuña.

There are five main components that describe a SPARQL query. First, **Prefix Declarations** serve as shortcuts for later in the query, similar to the Turtle shortcuts. Following prefixes, the **Dataset Clause** allows for specifying one or more RDF graphs over which the query should be executed. When no dataset clause is specified, query patterns are matched against the default graph which usually corresponds to all of the data loaded and indexed. The **Result Clause** indicates what type of query is being executed and what results are expected. The most relevant part is the **Query Clause**, where the query patterns to match against the data are specified and used to generate variable bindings. Finally, the **Solution Modifiers** permit to order, slice or paginate the results. Note that the only mandatory part is the result clause, though most queries also include at least one query clause. An example of a SPARQL query is represented in Listing 2.3, where we are looking for which awards people who were born in Vicuña have won.

For example, if the RDF triples shown in Listing 2.1 were contained in the RDF graph <http://dbpedia.org/data/Gabriela_Mistral.ttl>, the results expected from the SPARQL query in Listing 2.3 would be as shown in Table 2.1.

| ?people | ?award |
|---|---|
| dbr:Gabriela_Mistral | dbr:Nobel_Prize_in_Literature |
| dbr:Gabriela_Mistral | dbr:National_Prize_for_Literature_(Chile) |

Table 2.1: Results from SPARQL query example in Listing 2.3.

There are four types of SPARQL queries, which are defined by the first keyword used in the *Result Clause*. The example shown above is a SELECT query, which requests a list of bindings for variables specified in the *Query Clause*. Since a SELECT query returns duplicate results by default, it can include either the DISTINCT keyword to filter duplicate results, or the REDUCED keyword that may allow duplicates such that the engine can choose whatever it deems to be more efficient. An ASK query returns a boolean value indicating whether or not the query's results are non-empty. A CONSTRUCT query provides an RDF template with placeholders to be filled later, thus returning an RDF graph according to those inserted variables. The last type is the DESCRIBE query, which provides an RDF description for a particular RDF term. For this work we only focus on SELECT and ASK type queries.

Knowing how to define the content inside a *Query Clause* is crucial in order to specify what results a SPARQL query should return. There are many core features defined over the basic graph patterns that are used to create complex query patterns in query clauses. Among these patterns, we highlight the FILTER keyword that serves to establish conditions that a query solution should match. These conditions can be constructed from a broad arsenal of tools: arithmetic operators and comparators, built-in functions, casting, boolean connectives and even user-defined functions. We also mention the UNION operator, that allows joining results from two groups of query patterns. Another important feature is the OPTIONAL feature, which allows for matching data if available (if not, the corresponding result is still returned, where the variables exclusive to the OPTIONAL clause are left unbound).

After retrieving results from the *Query Clause*, such results can be divided or modified. The ORDER BY operation sorts results in ascending (ASC) or descending (DESC) order based on one or more variables; and the LIMIT keyword restricts the amount of results to return. Finally, the OFFSET clause allows the query to skip a certain number of results.

Besides the basic operations provided by the original SPARQL standard, the later update to SPARQL 1.1 [80] brought a wide-range of new features that increase the expressiveness and capabilities of this query language. Some core additions are property paths, aggregation, binding variables, subqueries, updates, new format outputs (CSV, TSV, JSON), among others. For this work we want to highlight the use of *property paths* and *aggregation* that will be often used. *Property paths* allow the user to match paths of arbitrary length in an RDF graph using regular expressions. *Aggregation* techniques include operations like count, max, min, sum, etc.; and can be applied over query results grouped by common terms.

Though many other SPARQL features are left to be mentioned, we focused on the features described above since those are the important ones for the development of this work.

### 2.1.4. Linked Open Data Cloud

Having briefly mentioned some core components of the Semantic Web, we have scratched just a tiny portion of what the Semantic Web concept means, and how it can be deployed on the Web. Most of what defines the use of the Semantic Web on the Web itself resides in understanding **Linked Data**. The early attempts to publish RDF on the Web tended to produce large dumps of data rarely interlinked with other RDF datasets and using different conventions. These issues ended up leaving entire isolated islands of RDF datasets that were difficult to access and with little chance of being discovered by other communities. Therefore, *Linked Data* emerged as a set of principles and best practices [17] to provide an environment where Semantic Web standards can be effectively deployed on the Web.

The four Linked Data principles arise from the Web Design Issues document published by Berners-Lee [17]. According to this document, these principles are: (1) use URIs as names for things, (2) use HTTP URIs so those names can be looked up, (3) return useful information upon lookup of those names, and last (4) include links by using URIs that dereference to remote documents.

Besides these principles, there is an emphasis on publishing data that can be easily processed, reused and exchanged between machines. As an approach to bootstrap Semantic Web publishing, a 5-star system [17] was promoted to describe the quality of published RDF data. Each star corresponds to one of the following considerations: (1) publish data under an open license, (2) publish structured data, (3) use non-proprietary formats, (4) use URIs to identify things, and (5) link the published data to other data.

A community project named *"Linking Open Data"*, supported by W3C and inspired by the growth in Open Data, emerged to promote these Linked Data principles [84]. The community project aims to introduce the benefits of Semantic Web technologies to the Open Data movement and to bootstrap the Web of Data by including many emerging open datasets. The community published guidelines are based on the core principles mentioned before. Among those guidelines, the main ones are:

- **Dereferencing practices**: describes how to identify and perform lookups for either entities or documents. This includes recommendations on indirect URIs to signify distinctions on a HTTP level or providing as detailed an RDF description as possible when dereferencing resources.

- **Linking Aliases**: allow the use of multiple URI aliases that refer to the same thing. For example, owl:sameAs links can be used to specify equivalence between resources in different datasets.

- **Describing Vocabularies Terms**: promotes the shared use of common vocabularies of class and property terms. A common example is to use FOAF to describe people.

- **Provision of SPARQL Endpoints**: though not required, providing a `SPARQL` endpoint for a given Linked Data site gives consumers a single-point-of-access to query over the merge of contributions on that site.

Based on these standards and guidelines, the **Linking Open Data Cloud**, a set of more than 300 different interlinked `RDF` datasets, has been constantly growing and including a wider range of topics. Some of the most relevant datasets are DBpedia [107], whose content is mostly based on Wikipedia articles, Freebase [24], a dataset previously supported by Google, and Wikidata [183], a large dataset supported by its community. Our work is mainly focused on the latter knowledge graph, which we will introduce in the next section.

### 2.1.5. Wikidata

As described by Vrandečić and Krötzsch [183], **Wikidata** is a free collaborative Knowledge Graph founded by the Wikimedia Foundation[1] in October 2012. Given the constant growth of its sister project Wikipedia, one of the most popular online encyclopedias, Wikidata is introduced as a new multilingual *"Wikipedia for data"*.

Despite Wikipedia's rich amount of data, consisting of more than 30 million articles in at least 287 languages, it started to face serious limitations in terms of providing data easily to the community [183]. There was the lack of direct access through query services or downloadable data exports, the same information often needed to be manually maintained in the same articles across many languages and across many articles within a single language, among others limitations. Wikidata aims to overcome many of Wikipedia's limitations by managing its data centrally.

Wikidata offers many features that make it an attractive and potentially useful resource for sharing information and connecting communities. Some aspects to consider about Wikidata are:

- **Openly editable**: allows any user to extend and edit the stored information.

- **Community control**: contributor community control that not only supervises the data being published but also the schema of the data.

- **Plurality**: though many facts can be disputed or be uncertain, Wikidata provides mechanisms to organize conflicting data for coexisting together.

- **Secondary data**: gathers facts published in primary sources including references to these sources.

[1] `https://wikimediafoundation.org/our-work/wikimedia-projects/`

- **Multilingual data**: all data have universal meaning and most of it is not tied to a single language. There is only one universal version of Wikidata.

- **Easy access**: data published under liberal legal terms are made easily accessible through web services, allowing the widest possible reuse.

- **Continuous evolution**: Wikidata grows with its community of editors and developers, so new features are constantly being added.

Wikidata has its own data model but further offers exports that follow the Semantic Web standards [49]. To identify items, Wikidata provides unique IDs, which are highly reusable and provide unambiguous definitions that do not depend on language labels. The same example of the entity Chile mentioned before is now represented in Wikidata as `http://www.wikidata.org/wiki/Q298`.

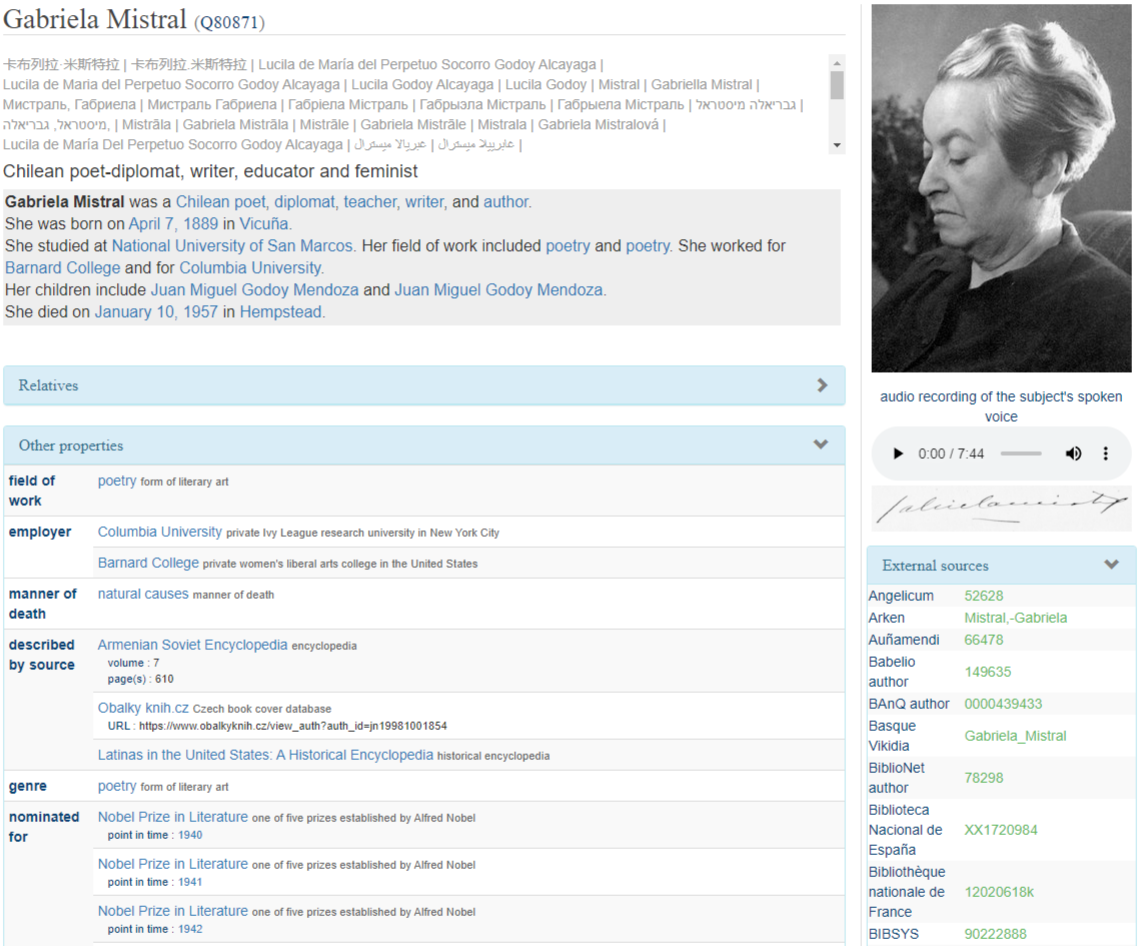
```
# PREFIX DECLARATIONS
@prefix wd: <http://wikidata.org/wiki/>
@prefix wdt: <http://wikidata.org/prop/direct/>
@prefix p: <http://wikidata.org/prop/>
@prefix pq: <http://wikidata.org/prop/qualifier/>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

# RDF TRIPLES
# Gabriela Mistral
wd:Q80871 rdfs:label "Gabriela Mistral"@en ;
    wdt:P106 wd:Q49757 ;
    wdt:P19 wd:Q201007 ;
    wdt:P166 wd:Q37922, wd:Q860699 .
# Vicuna
wd:Q201007 rdfs:label "Vicuña"@en ;
    p:P1082 _:population2017 ;
    wdt:P131 wd:Q721224 ;
    wdt:P17 dbr:Chile .
_:population2017 pq:P1082 25085
# Elqui Province
wd:Q721224 wdt:P131 wd:Q2121 .
```

Listing 2.4: Set of RDF triples about Gabriela Mistral in Wikidata.

Since some facts cannot be directly expressed by simply using the property-value convention, like the population of Chile across different years, Wikidata also provides additional subordinate property-value pairs called *"qualifiers"*. **Qualifiers** can be used to state contextual information such as validity time or ternary relations (e.g. cast members of a movie with their role). Another way to understand qualifiers is by looking at Wikipedia infoboxes which have a similar data representation.

[2] https://reasonator.toolforge.org/?&q=80871



Gabriela Mistral (Q80871)

卡布列拉·米斯特拉 | 卡布列拉.米斯特拉 | Lucila de María del Perpetuo Socorro Godoy Alcayaga | Lucila de Maria del Perpetuo Socorro Godoy Alcayaga | Lucila Godoy Alcayaga | Lucila Godoy | Mistral | Gabriella Mistral | Мистраль, Габриела | Мистраль Габриела | Габріела Містраль | Габрыэла Містраль | Габрыела Містраль | גבריאלה מיסטראל | מיסטראל, גבריאלה | Mistrāla | Gabriela Mistrāla | Mistrāle | Gabriela Mistrāle | Mistrala | Gabriela Mistralová | Lucila de María Del Perpetuo Socorro Godoy Alcayaga | غبريالا ميسترال | غابرييلا ميسترال |

Chilean poet-diplomat, writer, educator and feminist

**Gabriela Mistral** was a Chilean poet, diplomat, teacher, writer, and author.
She was born on April 7, 1889 in Vicuña.
She studied at National University of San Marcos. Her field of work included poetry and poetry. She worked for Barnard College and for Columbia University.
Her children include Juan Miguel Godoy Mendoza and Juan Miguel Godoy Mendoza.
She died on January 10, 1957 in Hempstead.

Relatives

Other properties

| | |
|---|---|
| **field of work** | poetry form of literary art |
| **employer** | Columbia University private Ivy League research university in New York City |
| | Barnard College private women's liberal arts college in the United States |
| **manner of death** | natural causes manner of death |
| **described by source** | Armenian Soviet Encyclopedia encyclopedia<br>volume : 7<br>page(s) : 610 |
| | Obalky knih.cz Czech book cover database<br>URL : https://www.obalkyknih.cz/view_auth?auth_id=jn19981001854 |
| | Latinas in the United States: A Historical Encyclopedia historical encyclopedia |
| **genre** | poetry form of literary art |
| **nominated for** | Nobel Prize in Literature one of five prizes established by Alfred Nobel<br>point in time : 1940 |
| | Nobel Prize in Literature one of five prizes established by Alfred Nobel<br>point in time : 1941 |
| | Nobel Prize in Literature one of five prizes established by Alfred Nobel<br>point in time : 1942 |

audio recording of the subject's spoken voice

0:00 / 7:44

External sources

| | |
|---|---|
| Angelicum | 52628 |
| Arken | Mistral,-Gabriela |
| Auñamendi | 66478 |
| Babelio author | 149635 |
| BAnQ author | 0000439433 |
| Basque Vikidia | Gabriela_Mistral |
| BiblioNet author | 78298 |
| Biblioteca Nacional de España | XX1720984 |
| Bibliothèque nationale de France | 12020618k |
| BIBSYS | 90222888 |

Figure 2.3: Example of an external application of Wikidata using Reasonator[2].

For a better understanding, let us recall the same data about Gabriela Mistral, but now using Wikidata resources, as shown in Listing 2.4. Since different knowledge graphs do not necessarily represent the same information in the same way, we can appreciate slight differences like how the city Vicuña is described. In this case, Vicuña is not directly represented as part of Coquimbo but instead only the Elqui Province. Nevertheless, both representations are intrinsically correct since both are portraying the same fact. Another difference is the use of a Wikidata property qualifier to denote that the population for Vicuña shown in Listing 2.4 is the population of 2017, in contrast to the DBpedia representation, shown in Listing 2.2, which does not show that fact.

Ultimately, the data in Wikidata lends itself to numerous applications on very different levels of data integration. Wikidata provides many language labels and descriptions for many terms in different languages, which allows any service to present and/or or translate information for various audiences. Some applications are built to access Wikidata's data more conveniently and effectively. One example is illustrated by Figure 2.3, showing information about Gabriela Mistral retrieved by the data browser *Reasonator* using the Wikidata API. Additionally applications can be enriched with information provided by Wikidata, such as how Google Maps uses Wikidata's geographical labels to enhance its application interface.

On a more advanced level, many research analyses can be performed over the information in Wikidata in order to derive new insights beyond its surface data. A couple of potential examples are analyses using logical reasoning to understand intrinsic meaningful relationships among entities, or statistical evaluations over the data to analyze biases like the ones that involve language coverage [78] or gender balance [184]. Wikidata is already an important platform and has the potential to be a major resource for both researchers and developers [29, 182].

## 2.2. Information Extraction

### 2.2.1. Information Extraction methods with Semantic Web technologies

As the Semantic Web aims to make structured data available enabling high levels of automation, there are still some challenges regarding the increasing demand for information. There is still a gap between the coverage of structured and unstructured data on the Web [145]. However, making high-quality annotations on unstructured data is not a trivial task because it requires processing vast amounts of information that is constantly changing.

Thus, automatic techniques for extracting and annotating information have gotten more attention in the context of the Semantic Web. As described by Martinez et. al., Information Extraction (IE) refers to the automatic extraction of implicit information from unstructured or semi-structured sources [124]. IE methods are used to identify entities, concepts and/or semantic relations implicit in an input source, typically a text in natural language.

Many systems have been developed to automate the extraction or enrichment of Semantic Web resources such as ontologies, knowledge graphs, etc. These systems are often based on Information Extraction methods which usually rely on techniques from areas such as Natural Language Processing, Machine Learning and Information Retrieval.

The combination of tools from the Semantic Web and Information Extraction areas presents two perspectives: using Information Extraction to populate the Semantic Web, or using Semantic Web resources to improve Information Extraction processes. In this work we focus on the last perspective mentioned, in particular how to extract and link entities over unstructured input sources (such as natural language questions).

An entity is understood as an atomic element within a Semantic Web Knowledge Graph or ontology. Entity Extraction & Linking (EEL) is then the task of identifying mentions in a text or document, and linking them as entities to one or more reference knowledge graphs. EEL is typically divided into two main steps: a recognition stage where relevant named entities are identified, and a disambiguation stage, where entities are mapped to candidate resources in the Knowledge Graph and subsequently ranked. Entity extraction often uses off-the-shelf Named Entity Recognition (NER) tools to recognise relevant entities. After extraction, Entity Linking follows, where the disambiguation of the spotted mentions links each mention to an identifier in a target knowledge graph, and may include a score or weight calculation that denotes the confidence or support over the output annotations. We now discuss the Entity Linking process in more detail.

### 2.2.2. Entity Linking

Entity Linking is the task of linking mentions in text to their corresponding entities in a Knowledge Graph (e.g. Wikipedia, Wikidata, DBpedia) [189]. Aside from extracting entities from a knowledge graph, a disambiguation step is also needed. For example, for the question *"Has Claudio Bravo played for Manchester City FC?"*, Entity Linking with respect to Wikidata should link the mention *"Claudio Bravo"* to the Chilean football goalkeeper Claudio Bravo (`Q313161`) instead of the Chilean painter Claudio Bravo (`Q491787`), given the context of the sentence (a person playing in a football club). Some applications of Entity Linking involve fields such as information retrieval [37, 22, 25] knowledge fusion [45, 23], or Knowledge Graph population [150, 46, 131].

Commonly, the entity linking process is divided into three modules: candidate entity generation, candidate entity disambiguation and linking the result. A formal description of Entity Linking according to Wu and He [189] is the following: Given a set of documents $\mathrm{d} = \{\mathrm{d}_1, \mathrm{d}_2, \ldots\}$ and a Knowledge Graph $K$, we can get a mention set $M = \{m_1, m_2, \ldots\}$ using a Named Entity Recognition tool. For each $m_\mathrm{i} \in M$, we can get a candidate set $C = \{c_1, c_2, \ldots\}$ from a knowledge base. The goal of Entity Linking is to choose an entity from $c \in C$ for each mention $m \in M$. If $score(m, c)$ is below $\tau$ ($\tau$ is a threshold) for all $c \in C$, then the target entity of m is *Not In Lexicon* (NIL); otherwise, $m$ will be linked to $c'$ such that $score(m, c') = max\{score(m, c) \mid c \in C\}$. Figure 2.4 shows a general model including each phase and the formal description mentioned.

The first module selects the candidate entities for each mention identified in the text and finds related entities in the knowledge base. For example, *Claudio Bravo* is related to the entities *Claudio Bravo (football goalkeeper)* and *Claudio Bravo (painter).*

The second module ranks the candidate entities by combining different features of entities and assigning scores to each candidate. Some features could be entity popularity, entity type, similarity between names or context in the query, topic similarity and a combination of several features. In the example, *Claudio Bravo (football goalkeeper)* should have a higher score than *Claudio Bravo (painter)* due to the football-related context of the sentence.

The last module selects the target entity for each mention according to the ranking derived

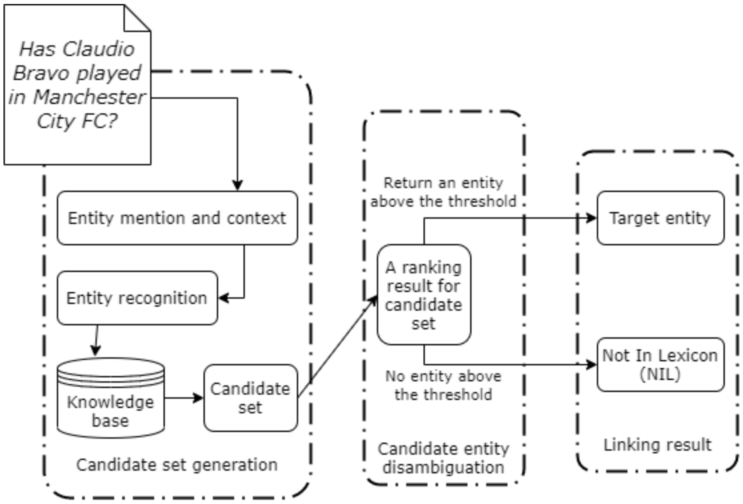

Has Claudio Bravo played in Manchester City FC?
Entity mention and context
Entity recognition
Knowledge base
Candidate set
Candidate set generation
Return an entity above the threshold
A ranking result for candidate set
No entity above the threshold
Candidate entity disambiguation
Target entity
Not In Lexicon (NIL)
Linking result

Figure 2.4: A general model of Entity Linking based on Wu and He [189]

from the previous module. The candidates with the highest score per mention are selected among the candidates whose scores are above the threshold. If scores from all candidates are below the threshold, some systems return a NIL clustering [94], although we will work with systems that directly discard results that do not satisfy the threshold.

There are various existing methods for addressing candidate entity generation and disambiguation. The methods for candidate entity generation can be divided into methods based on dictionaries [203, 79], direct search [126, 46] and probabilistic methods [60, 138]. On the other hand, the methods for candidate entity disambiguation can be divided into methods based on similarity computation [38, 28], machine learning [60, 203] and graphs [70, 79].

One of the main difficulties in Entity Linking is the high ambiguity of entity mentions, which makes it more difficult to understand the meaning of entity mentions. These ambiguities include polysemies, which refer to mentions that correspond to many entities (e.g. *Claudio Bravo*) or multiword synonyms, which refer to entities that may have many kinds of surface forms (e.g. *Manchester City FC* is also known as *The Citizen* or *The Sky Blues*). Another problem happens when selecting entities in the linking results phase since the threshold is selected manually and can lead to the problem that correct targets can be below the threshold, thus being discarded.

Entity Linking evaluation criteria are usually based on Precision, Recall and F1-score. For each of these metrics there is a micro measure and a macro measure [36]. The macro measure gives equal importance to each document since it first calculates the relevant measure over each document, and then calculates the arithmetic average. On the other hand, the micro measure considers all mentions as part of one document when calculating the relevant measure, thus giving more importance to documents with more mentions. The following equations represent the micro and macro measures of Precision and Recall:

$$
\begin{aligned}
Precision_{micro} &= \frac{|S \cap G|}{|S|} \\
Recall_{micro} &= \frac{|S \cap G|}{|G|} \\
Precision_{macro} &= \frac{\sum_{\mathrm{i}=1}^{|D|} \frac{|s_{\mathrm{i}} \cap g_{\mathrm{i}}|}{|s_{\mathrm{i}}|}}{|D|} \\
Recall_{macro} &= \frac{\sum_{\mathrm{i}=1}^{|D|} \frac{|s_{\mathrm{i}} \cap g_{\mathrm{i}}|}{|g_{\mathrm{i}}|}}{|D|}
\end{aligned}
$$

where $D$ represents a document containing a number of texts, $G$ is the set of annotated entities that should be linked in a document ($g_{\mathrm{i}}$ is the equivalent for each document), $S$ the set of linked entities generated by a system in a document ($s_{\mathrm{i}}$ is the equivalent for each document). The Precision is the ratio of entities correctly linked to a Knowledge Graph over the linked entities generated by a system, while the Recall is the ratio of entities correctly linked to a Knowledge Graph over the entities that should be correctly linked. Then, the F1-score is a measure that combines Precision and Recall as two interacting values and is calculated with the following formula (based on the harmonic mean of both measures):

$$F1_x = 2\frac{Precision_x \cdot Recall_x}{Precision_x + Recall_x}$$

where $x$ corresponds to the micro or macro version of the F1-score. Aside from Precision, Recall and F1-score, some systems also include an Accuracy measure that includes NIL entities, though we will not consider this metric in our evaluations as NIL entities in questions cannot generate results for queries.

There are many datasets used for evaluation such as KORE50 [88], AIDA-CoNLL [89], NEEL [153], and OKE2016 [144], which can be evaluated over knowledge bases such as Wikipedia, DBpedia [107], and YAGO [166]. To the best of our knowledge, there is no dataset for evaluating Entity Linking over Wikidata, but since Wikidata, DBpedia and Wikipedia are all interlinked, we can use datasets with labels for any such resource.

In this work we will use several Entity Linking systems that we will briefly describe later. The criteria to choose these systems were:

1. have a public API available that allows at least 10,000 requests per day;
2. have references/papers explaining how the system functions; and
3. work over either Wikipedia, Wikidata, DBpedia or YAGO.

Given these criteria, the selected systems are: *DBpedia Spotlight*, *AIDA*, *TAGME* and *OpenTapioca*.

### 2.2.2.1. DBpedia Spotlight

DBpedia Spotlight [128] is a system that automatically annotates text documents with DBpedia URIs. The system allows users to configure annotations to their specific needs through the DBpedia Ontology and quality measures provided by the system. Their approach is divided into four phases.

The **spotting stage** identifies the phrases in a sentence that may contain a mention of a DBpedia resource. Before performing the spotting process, the system builds a lexicon of labels extracted using a graph of labels, redirects and disambiguation pages in DBpedia. The labels of DBpedia resources are created from Wikipedia page titles, which are seen as community-approved surface forms. Redirects to URIs indicate synonyms or alternative surface forms (including common misspellings and acronyms) whose labels also become surface forms. Disambiguation pages provide links from ambiguous surface forms to the resources they potentially link to. The resulting collection of surface forms composes the set of labels for the target resources.

As an additional resource for the later disambiguation stage, a collection of occurrences for each resource based on wikilinks (page links in Wikipedia associated with one resource) is stored as a document in a Lucene[3] index.

A **candidate selection** is then employed to map resource names to candidate disambiguations spotted in the previous phase. The DBpedia Lexicalization dataset is used to determine candidate disambiguations for each surface form. This phase aims to reduce the number of disambiguation possibilities keeping a trade-off between time performance and system Recall.

Following candidate selection is the **disambiguation stage**, where the system uses the context gathered from surface forms to choose the best choice amongst candidates. DBpedia resource occurrences are modeled in a Vector Space Model (VSM) [156] where each DBpedia resource is a point in a multidimensional space of words. The *Term Frequency* (TF) weight and the *Inverse Candidate Frequency* (ICF) weight are used to score each candidate.

The TF weight represents the relevance of a word for a given resource. The ICF weight is proposed given that the standard *Inverse Document Frequency* (IDF) weight [98] only identifies the global importance for a word, and thus fails to capture the importance of a word for a specific set of candidate resources. Instead, the ICF weight aims to weight words based on their ability to distinguish between candidates for a given surface form. The intuition behind the ICF formula is that the discriminative power of a word is inversely

[3] `http://lucene.apache.org`

proportional to the number of DBpedia resources it is associated with.

Having the VSM representation of DBpedia resources with $TF * ICF$ weights, the disambiguation process is performed by ranking candidate resources using the cosine similarity score between their context vectors and the context surrounding the surface form.

Finally, annotations can be customized through **configuration parameters** in order to tune parameters to a specific task. The offered configuration parameters can be used to allow/deny URIs with some classes or its subclasses, set a required minimum of inlinks, establish thresholds to prioritize resources relevant to the topic, reduce highly ambiguous resources, and configure a disambiguation confidence to keep a good trade-off between avoiding incorrect annotations and losing correct annotations.

A web-service[4] is available for integration with external web processes. The service is implemented through RESTful and SOAP web services for the annotation and disambiguation processes, and supports various output formats (HTML, XML, JSON or XHTML+RDFa).

#### 2.2.2.2. AIDA

The Accurate Online Disambiguation of Named Entities (AIDA) [89, 198] is an online tool that performs entity detection and disambiguation over the YAGO Knowledge Graph. The system's approach combines the use of a Named Entity Recognition (NER) tool with a graph-based mapping.

The system automatically detects mentions using the Stanford NER Tagger[5] based on a Conditional Random Fields (CRF) Classifier [56]. Their approach uses Gibbs sampling to identify non-local structures while preserving tractable inference by simulated annealing in place of Viterbi decoding in sequence models. They use this technique to improve an existing CRF-based information extraction system with long-distance dependency models.

The collection mapping relies on a graph constructed with mentions and their candidate entities as nodes and two types of edges: *mention-entity edges* and *entity-entity edges*. The **mention-entity edges** are weighted edges between mentions and their candidate entities (one edge per candidate) and represent the similarity between the context of each node. The **entity-entity edges** are weighted edges between different entities and represent the coherence, i.e. the semantic relatedness between both nodes.

The *similarity* between a mention and a candidate entity is defined as the linear combination of the *prominence* of an entity and the *context similarity* between a mention and a candidate entity. The **prominence** (or popularity) of an entity is calculated using the frequency of Wikipedia-based link href anchor texts and links referencing the entity. The **context similarity** is calculated in different ways for a mention's context and an entity's

[4] `https://www.dbpedia-spotlight.org/`

[5] `https://nlp.stanford.edu/software/CRF-NER.shtml`

context.

The context of a mention simply considers all tokens in the document as the context [171]. For the context of an entity, the system considers entity keyphrases, which are pre-computed phrases derived from link anchors in Wikipedia articles that entities connect to [171]. These include phrases in the entity article that contain category names, citation titles, external references or titles of incoming links. This process forms a keyphrase set $KP(\mathrm{e})$ for each entity e.

The *Mutual Information* (MI) measure, which quantifies the "amount of information" one random variable obtains from observing another one, is used to quantify the specificity weight of a keyword with regard to an entity. In this context, the MI for each keyword $w$ is based on a joint probability $P(\mathrm{e}, w)$ that reflects the probability of $w$ to be contained in either the keyphrase $KP(\mathrm{e})$, or any of the keyphrase sets of entities linked to e.

Since keyphrases can rarely match multi-word keyphrases (e.g. the phrase *"Nobel Prize winner"* may occur in the form of *"Nobel winner"*), a partial-match model is added to improve coverage [170]. This model matches individual words and rewards their proximity. Following this approach, a phrase's cover is computed for each keyphrase, which consists of the shortest window of words that maximize the number of words of the keyphrase. For example, the text *"winner of many prizes including a Nobel"* the cover length of the keyphrase *"Nobel award winner"* is 7. Then, the partial-match score of a phrase in a text is calculated using the MI weights of the keyphrase words in the phrase and the ones included on the phrase's cover.

Lastly, the similarity score between a mention and a candidate entity is computed by summing all the partial matching scores of the phrases that are part of the keyphrase $KP(\mathrm{e})$.

The **coherence** weight between a pair of entities is calculated using the number of incoming Wikipedia links that both entities share in their Wikipedia articles, which are denoted using crossreferencing properties such as *same-as*. This approach is polished up by considering the total number of entities in the Wikipedia collection [130].

Having built the weighted graph, the system aims to provide an output graph which consists of the graph reduced to a dense subgraph where each mention is connected to only one candidate entity. The concept of density refers to the minimum weighted degree in the subgraph. The calculation of this subgraph is computed using a greedy algorithm where its main loop performs two main steps in each iteration: (1) identify the entity node with the lowest weighted degree and (2) remove this node and its incident edges only if it is not the last remaining candidate entity for one of the mentions.

AIDA provides a HTTP JSON web service[6] for annotating texts. It can be accessed via CURL requests and only needs to be provided by the text that needs to be annotated.

[6] `https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/ambiverse-nlu/aida`

#### 2.2.2.3. TAGME

The TAGME system permits augmenting plain-text with hyperlinks to Wikipedia pages [52]. It can work over short and poorly composed texts. The system annotation is divided into three phases.

The first stage includes generating an anchor dictionary based on Wikidata pages. An anchor (also referred to as a spot) for a page $p$ is defined as the text used in the hyperlink of another page to refer to $p$. These anchors are extracted from Wikipedia pages and also include titles of redirect pages and other variants [39]. The anchors composed by one character or just numbers and anchors with low-frequency are removed. The set of all Wikipedia pages linked by a given anchor $a$ is denoted as $Pg(a)$. The final anchor dictionary is indexed using Lucene.

Then, each annotation of an anchor $a$ with some page $p_a \in Pg(a)$, denoted $a \rightarrow p_a$, forms what the authors refer to as a *sense*[7]. These senses are built using all Wikipedia pages but discarding disambiguation pages, list pages and redirect pages, also indexed by Lucene. These results are indexed in a link-graph using Webgraph[8].

Often an anchor $a$ has more than one candidate sense, so a disambiguation process is performed. Having the set of all anchors contained in a text $T$, denoted by $A_T$, the system tries to disambiguate each anchor $a \in A_T$ by computing a score for each possible sense $p_a \in Pg(a)$. This process is implemented using a voting scheme that computes for each other anchor $b \in A_T \backslash \{a\}$ its vote for the annotation $a \rightarrow p_a$.

To disambiguate the anchor $a$, a ranking process is designed to select its best annotation $a \rightarrow p_a$. The TAGME system presents two variants for the ranking algorithm: one based on a classifier (DC) and the other based on a threshold (DT). DC uses as features the score $rel_a(p_a)$ and the commonness $Pr(p_a|a)$ to train a classifier to calculate the *"probability of correct disambiguation"* for all senses $p_a \in Pg(a)$. The $p_a$ reporting the highest classification score is selected. The other approach, DT, computes the top-$\varepsilon$ best senses $p'_a \in Pg(a)$ according to their $rel_a(p'_a)$ score and then annotates $a$ with the sense that obtains the highest commonness amongst them. For time performance optimization, both variants discard senses with a commonness below a certain threshold $\tau$.

Finally, in the anchor pruning stage, the set $M(A_T)$ of candidates produced in the previous stage are pruned to avoid meaningless anchors. A *bad anchor* is defined by a score computed using two features: the link probability $lp(a)$ of the anchor $a$ and the coherence of its candidate senses with respect to the candidate senses of other anchors in $M(A_T)$ [130]. This link probability $lp(a)$ corresponds to the ratio between the number of times the phrase $a$ occurs

[7] For example, a mention of Gabriela Mistral on any given Wikipedia article, *"Gabriela Mistral"* is the anchor, and two possible senses are `https://en.wikipedia.org/wiki/Gabriela_Mistral` or `https://en.wikipedia.org/wiki/Museo_Gabriela_Mistral`.

[8] `http://webgraph.dsi.unimi.it`

as an anchor in Wikipedia, and the frequency with which this phrase occurs in both anchor and non-anchor occurrences.

The *coherence* between two candidate annotations is equivalent to the process followed in the AIDA system. A score $\rho(a \to p)$ is then calculated per candidate annotation, which is compared to a threshold $\rho_{NA}$, so annotations with a score lower than the given threshold are pruned by setting $a \to NA$ (a fake page to denote pruned annotations). Two approaches that combine $lp$ and coherence are presented to compute this score: one is an average between the two features and the second is a linear combination trained via linear regression.

A web-service hosted by D3Science Infrastructure[9] is available to annotate text. The system allows to set a threshold $p$ used for discarding annotations.

#### 2.2.2.4. OpenTapioca

OpenTapioca is a lightweight Named Entity Linking system that works over Wikidata [42], and is restricted to people, locations and organizations. Let $D$ be a document; a spot $s$ is a pair of start and end positions in $D$. This spot defines a phrase $\mathrm{d}_s$ in $D$, and a set of candidate Wikidata entities $E_s$. The OpenTapioca system is based on a binary classifier that predicts for each spot $s$, and each candidate entity e linked to that spot, if $s$ should be linked to e. This approach combines *local compatibility* and *semantic similarity* to classify entities according to their context.

The **local compatibility** for an entity e with a phrase $\mathrm{d}_s$ is represented by a vector of features that considers the popularity of the entity and the commonness of the phrase. The popularity of an entity is estimated by a log-linear combination of its number of statements $n_{\mathrm{e}}$, site links $s_{\mathrm{e}}$ and its PageRank $r(\mathrm{e})$ (calculated using Wikidata statement values and qualifiers as edges). The commonness of a phrase is estimated using a unigram language model trained from Wikidata item labels.

Since the aforementioned features do not consider the context of the mention, a graph is defined whose nodes are the candidate entities and edges link semantically related entities. The approach consists of finding a combination of candidate entities which are both highly compatible and densely related in the graph.

Along these lines, the **semantic similarity** measure is used to make the process context-sensitive. An adaptation of the Han et al.'s [79] approach is proposed, where a similarity metric $sim(\mathrm{e}, \mathrm{e}')$ is defined for each pair of entities $(\mathrm{e}, \mathrm{e}')$ that defines the probability that two random walks starting from e and e$'$ end up on the same item.

Next, a weighted graph $G_D$ is built where each vertex is a pair $(s, \mathrm{e})$ such that $\mathrm{d}_s \in D$ and $\mathrm{e} \in E_s$. A maximum distance $\rho_{max}$ is fixed for edges so a pair of vertices can only be linked if its distance is less than or equal to $\rho_{max}$, and both vertices are referring to a different

[9] https://sobigdata.d4science.org/web/tagme/tagme-help

mention. The weight is defined for each edge, which is proportional to the smoothed similarity between entities, discounted by the distance between mentions. The weighted graph $G_D$ is represented by a column-stochastic matrix $M_D$ which is an adjacency matrix normalized by its columns to sum to one.

The resulting matrix $M_D$ defines a Markov chain on the candidate entities that is used to propagate the local evidence, which helps to classify entities according to the context. A Markov chain is a mathematical model used to model transitions from one state to another, usually in a stochastic way. One particularity of Markov chains is that the stochastic process is "memoryless". That is, the probability of transitioning to any particular state depends solely on the current state and time elapsed[10].

Then, instead of combining the local features into a local evidence score as done by Han et al. [79], each local compatibility feature is propagated independently along the Markov chain. This allows for recording the features at each step, which defines a vector of features more sensitive to the context while keeping the number of features small. Finally, a linear support vector classifier is trained on these features, which defines the final score of each candidate entity. For each spot, the system picks the highest scoring candidate that the classifier predicts, if any.

OpenTapioca is available through a web-service[11] implemented using Solr[12] and some Python libraries. Its service keeps synchronized with Wikidata in real time.

### 2.2.3. Sequence Labeling

Another application of Information Extraction methods is Sequence Labeling [72, 117], also known as Semantic Role Labeling [65]. Sequence Labeling is a semantic analysis tool that can be used to detect meaningful entities, relationships or semantic properties in a given sentence. For example, for the sentence *"Barbara lives in Santiago"*, the name *Barbara* could be identified as the subject of the sentence or as a person, while the name *Santiago* could be recognized as the object of the sentence or as a location.

Some traditional sequence labeling models are based on linear statistical models such as Hidden Markov Models [151] or Conditional Random Fields (CRF) [140, 114], which mainly rely on hand-crafted features and thus are difficult to adapt to new tasks or new domains [118]. On the other hand, recent works that combine Neural Networks and Word Embeddings have been broadly used to enhance sequential data modeling [34, 63]. The combination of these two components have shown better results on many Natural Languages Processing (NLP) tasks such as POS tagging [117, 93], Named Entity Recognition [33, 92] or

[10] One example of a Markov chain process is the probability question of getting a certain color ball from a bag of balls, when replacement is allowed each time a ball is drawn.

[11] `https://opentapioca.org/`

[12] `https://lucene.apache.org/solr/`

Speech Recognition [75], mainly due to its capacity to learn and generalize with information learned from unlabeled data, which reduces the ambiguity issues that previous statistical models suffer from [118].

In particular, we are interested in Sequence Tagger systems which, given a sequence of words, provide a semantic meaning to words or composed words in the context of a given corpus. This semantic meaning varies depending on the task. For example, in Part-of-Speech (POS) tagging, words in a sentence are usually tagged as nouns, verbs, adjectives, adverbs, etc. Another area of use is Name Entity Recognition (NER), where the sequence tagger can identify entity names and tag them as person, location, organization, etc.

Then, the information output by the tagger can be used in other IE methods such as Entity Linking. The task fulfilled by the tagger can be adapted depending on the labels used (e.g. provide more entity types for NER) but the process of training and learning such labels should not change. We will apply these methods later to identify which sequences of terms in the question text refer to which elements of the query. The labels output by the system are usually known as BIO labels (where BIO means beginning-inside-outside) and demark a tag for a word and whether a word is the beginning of a tag, an inside part of a tag, or a word outside a tag. An example of a BIO label output can be seen in Figure 2.5.

The performance of a Sequence Tagger is measured using a per-word accuracy [122]. Let $W = (w_0, \ldots, w_n)$ denote a sequence of words, $(l_0, \ldots, l_n)$ a sequence of expected BIO labels and $(l'_0, \ldots, l'_n)$ a sequence of BIO labels output by a Sequence Tagger; a word $w_{\mathrm{i}}$ would be labeled correctly if its expected label $l_{\mathrm{i}}$ matches with the label $l'_{\mathrm{i}}$ delivered by the Sequence Tagger. Then, the Accuracy over $W$ will be $accuracy(W) = \frac{\text{\#words correctly labeled}}{\text{total \# words}}$.

In this work, we will use the Flair Framework [4], which provides up-to-date state-of-the-art language models and word embeddings in a simple interface. Flair is implemented in Python using the Pytorch framework for implementing Neural Network based models. One of the tools Flair provides is a Sequence Tagger model, which includes pre-trained models or the capacity of training a new model by providing training data. In order to understand how the Flair Sequence Tagger works, we will explain its two main components: Contextual String Embeddings [5] and its main architecture based on a Bidirectional LSTM-CRF model.

#### 2.2.3.1. Contextual String Embeddings

The construction of Contextual String Embeddings (aka Flair embeddings) is based on two Language Models (LM) where each one captures semantic-syntactic information for each word in the sentence; one model captures information from the “past” and the other model from the “future” of each word. The information from both models is combined to construct representations of words based on their surrounding context.

#### 2.2.3.1.1. Language Models

A Language Model (LM) is a probability distribution over sequences of words [146]. Language models can be character-level or word-level, where the difference lies in the atomic unit selected for the language model [146]. In this work we focus on character-level LMs.

For a character-level LM, the goal is to predict the expected character given a set of characters as context, i.e., to provide a good distribution $P(x_{0:T})$ over sequences of characters $(x_0, x_1, \ldots, x_T)$ [73]. Intuitively speaking, we aim to respond to the question: *given a certain sequence of characters, what is the most probable character that follows to that sequence?* (e.g. if we provide with the sequence "goalkeepe", it is very likely that the character 'r' should come next). Then, if the objective is to predict the next character given the previous characters, we will train a language a model to learn $P(x_t|x_0, ..., x_{t-1})$, for $0 < t <= T$, which gives an estimate of the predictive distribution over the next character given the previous characters.

Recent work on language models prefers architectures based on Recurrent Neural Networks [5]. Language models based on Neural Networks, also known as Neural Language Models, have shown better results than statistical language models [73], mainly because statistical models decrease their performance when dealing with large vocabulary size, which causes a data sparsity problem [48] (therefore their capacity to generalize is limited).

Among many applications of neural language models, a key application is the creation of word embeddings, which are vector representations of words typically based on the trainable weights within the Neural Networks used in the language model. Word embeddings have shown a great capacity of encapsulating semantic attributes of abstract concepts (e.g. we can capture to a certain extent that "red" is a color, because we identify a degree of semantic similarity when comparing its numeric values to other colors' embeddings), and such knowledge can be transferred to other tasks in order to enhance other models' learning capabilities (as we will do here for the Sequence Labeling task). In this case, the architecture used to construct Flair embeddings is the LSTM variant [87, 73, 201] which enhances the ability to encode long-term dependencies with its hidden states (weights in an LSTM architecture).

#### 2.2.3.1.2. Extracting Flair Embeddings

As mentioned earlier, the goal of a character-level language model is to estimate the distribution $P(x_{0:T})$ over a sequence of characters $x_0 : T = (x_0, x_1, \ldots, x_T)$. This results in a joint distribution over the entire sentence, which is the product of the conditional probability $P(x_t|x_0, ..., x_{t-1})$ of each character of the sentence:

$$P(x_0) = \prod_{t=0}^{T} P(x_T|x_{0:t-1}) \tag{2.1}$$

In an LSTM architecture, the conditional probability $P(x_\mathrm{i}|x_0, \ldots, x_{\mathrm{i}-1})$ is approximated as a function of the network output $h_t$:

$$P(x_T|x_{0:t-1}) \approx \prod_{t=0}^{T} P(x_T|h_t; \theta)$$

where $h_t$ represents the past context of the character sequence and is computed recursively using an additional memory cell $c_t$:

$$\begin{aligned} h_t(x_{0:t-1}) &= f_h(x_{t-1}, h_{t-1}, c_{t-1}; \theta) \\ c_t(x_{0:t-1}) &= f_c(x_{t-1}, h_{t-1}, c_{t-1}; \theta) \end{aligned}$$

where $\theta$ denotes all the parameters of the model. The proposed model includes a fully connected softmax layer on top of $h_t$, so the likelihood of every character is given by:

$$P(x_t|h_t; V) = \frac{\mathrm{e}xp(Vh_t + b)}{\|\mathrm{e}xp(Vh_t + b)\|}$$

where $h_t^b$ is denoted as the hidden states of the backward model calculated the same way as Equation 2.1. We also denote $h_t^f$ as the hidden states $h_t$ of the forward model.

Finally, output hidden states from both models are concatenated to form the final embedding which represents the surrounding context of a word itself. Formally, the contextual string embedding of a word-string that begins at character inputs with indices $t_0, t_1, \ldots, t_n$ is defined as:

$$w_\mathrm{i}^{CharLM} := \begin{bmatrix} h^f_{t_{\mathrm{i}+1}-1} \\ h^b_{t_\mathrm{i}-1} \end{bmatrix}$$

The result are Contextual String Embeddings capable of producing different representations for the same lexical word string in different contexts while capturing the semantics of contextual use together with the word itself. These embeddings are then used to boost standard Sequence Labeling models as explained next.

#### 2.2.3.2. Sequence Labeling Architecture

Though the Flair framework supports various sequence tagging models, the default is to use the model based on a Bidirectional LSTM with a Conditional Random Field layer on top of the final BiLSTM layer, also known as BiLSTM-CRF [93]. Let us denote by $w_o, w_1, \ldots, w_n$ the input of the model; then we have that:

$$r_{\mathrm{i}} := \begin{bmatrix} r_{\mathrm{i}}^{f} \\ r_{\mathrm{i}}^{b} \end{bmatrix}$$

where $r_{\mathrm{i}}^{f}$ and $r_{\mathrm{i}}^{b}$ are the forward and backward output states of the model. The final sequence probability is then given by a CRF over the possible labels $y$:

$$\widehat{P}(y_{0:n}|r_{0:n}) \propto \prod_{\mathrm{i}=1}^{n} exp(W_{(y_{\mathrm{i}-1},y_{\mathrm{i}})}r_{\mathrm{i}} + b_{(y_{\mathrm{i}-1},y_{\mathrm{i}})})$$

The component that improves the performance of these BiLSTM-CRF models is the addition of stacked embeddings, which are a combination of different types of embeddings. The way to combine each embedding vector is by concatenating them to form a final word vector representation. The final word representation chosen in this work is given by:

$$w_{\mathrm{i}} := \begin{bmatrix} w_{\mathrm{i}}^{CharLM} \\ w_{\mathrm{i}}^{GloV\mathrm{e}} \end{bmatrix} \tag{2.2}$$

where $w_{\mathrm{i}}^{GloV\mathrm{e}}$ is a precomputed GloVe embedding [141].

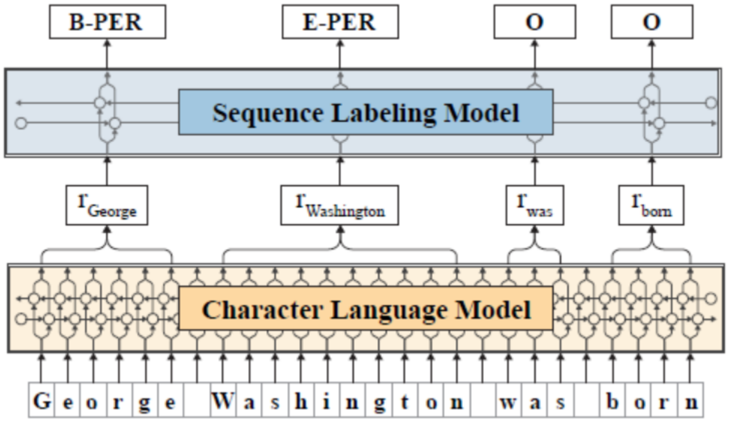


Figure 2.5: Flair Sequence Labeling architecture [4].

Note that these word representations are the input words for the BiLSTM-CRF. An example is shown in Figure 2.5, where the Character Language Model mentioned before is fed the sentence *"George Washington was born"* as a sequence of characters. The output delivered by the Language Model corresponds to the vector representation of each word following Equation 2.2. Then, these sequences of words are taken by the Sequence Labeling Model, which gives an output of a sequence of BIO labels such as to indicate that the phrase *"George Washington"* is tagged as a person (PER).

## 2.3. Semantic Parsing

As mentioned by Kamath and Das [100], Semantic Parsing is defined as the mapping from a natural language utterance into a semantic representation. These representations usually refer to logical forms, meaning representations or programs, which are executed over an underlying context such as relational tables or Knowledge Graphs. This execution yields a desired output like an answer to a question. For example, given a question in natural language, a semantic parser can aim to generate a valid `SPARQL` query based on the Wikidata's ontology grammar that produces the correct answer when executed over a Wikidata endpoint.

The first component of a Semantic Parsing framework is the **language** to represent logical forms or meaning representations such as logic based formalisms [109, 10], graph based formalisms [12, 135] or programming languages [54]. In particular, we focus on query languages such as `SQL` or `SPARQL`. Another component is the **grammar**, which is a set of rules used to decide the expressivity of a semantic parser. An example is the Combinatory Categorial Grammar for complex structured queries [165]. A last component is the **underlying context**, which is the environment over which the output mappings are interpreted or executed. Knowledge Graphs such as Wikidata or DBpedia serve as examples of an underlying context.

The early attempts for Semantic Parsing were systems based on rules or statistical techniques. Among the rule-based systems, systems could be based on pattern matching [97] or syntax-based systems [188]. Though their implementation is simple, rule-based systems tend to be domain specific, thus hard to adapt to other domains. On the other hand, statistical models are able to train given examples of input-output pairs from any domain. Many approaches require a lexicon as a-priori knowledge [202, 173], which is used to extract relevant semantic or syntactic information. Since these examples are usually manually annotated or require complex annotations, statistical models are hard to scale. There is also an issue with data sparsity, so these models only work in narrow domains.

Some of the most recent approaches that have emerged are based on Sequence-to-Sequence (Seq2seq) models, which usually uses an encoder-decoder framework based on neural networks. Some approaches implement an end-to-end paradigm where an intermediate representation is not needed to deliver a meaning representation; thus they do not rely on lexicons, templates or manually generated features. Though traditional approaches are able to better model and leverage the in-built knowledge of logic compositionality, approaches based on sequence models outperform traditional approaches due to the fact that Seq2seq-based models generalize better with more complex and longer sentences [95]. Furthermore, Seq2seq-based models can also generalize across domains [100].

In the following subsections, we discuss in more depth how systems based on Sequence-to-Sequence models work. First, we will briefly explain Sequence-to-Sequence models along with the approach we will use in this work: the Convolutional Sequence-to-Sequence model. Then, we will introduce Neural Machine Translation systems and how these models can be used for the task of translating natural language questions to `SPARQL` queries.

### 2.3.1. Sequence to Sequence models

The Sequence-to-Sequence (Seq2seq) model was first introduced by Cho et al. [34] for statistical Machine Translation. They proposed a neural network model based on an encoder-decoder framework which is based on recurrent neural networks (RNNs) [186, 155, 87]. More details about RNNs can be found in Appendix A.

In a Seq2seq architecture, the encoder converts a variable-length sequence into a fixed-length vector representation (i.e., it encodes the input sequence into a context vector) which is passed through to the decoder that transforms this fixed-length vector representation back into a variable-length sequence (i.e. decodes a context vector back to another output sequence). Figure 2.6 illustrates graphically how a Seq2seq looks, where the length $T$ of the input sequence does not necessarily equal the length $T'$ of the output sequence.

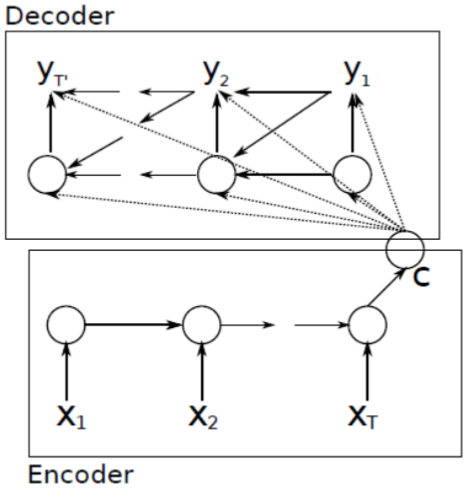


Figure 2.6: Sequence to Sequence model [72].

Technically, the model is learning a conditional distribution over a variable-length sequence conditioned on yet another variable-length sequence $p(y_1, \ldots, y'_T | x_1, \ldots, x_T)$. The encoder is an RNN that reads each symbol of an input sequence $x$ sequentially. While it is reading the current symbol on each step $t$, the hidden state $h_t^{\mathrm{e}}$ of the RNN changes are described as:

$$h_t^{\mathrm{e}} = f(h_{t-1}^{\mathrm{e}}, x_t)$$

After reading the end of the sequence, the hidden state of the RNN is the summary $c$ of the whole input sequence, also known as its context vector. Then, the decoder is another RNN trained to generate the output sequence by predicting the next symbol $y_t$ given the hidden state $h_t^{\mathrm{d}}$. This prediction is also conditioned on the previous predicted symbol $y_{t-1}$ and on the context vector $c$. Then, the hidden state of the decoder is defined for the step $t$, where $f$ is usually the *sigmoid* function:

$$h_t^{\mathrm{d}} = f(h_{t-1}^{\mathrm{d}}, y_{t-1}, c)$$

Similarly, the conditional distribution of the next symbol, where $g$ is commonly a *softmax* function since a valid probability must be produced, is defined as follows:

$$P(y_t|y_{t-1}, y_{t-2}, \ldots, y_1, c) = g(h_{t-1}^{\mathrm{d}}, y_{t-1}, c)$$

Both components of the sequence model are jointly trained to maximize the following conditional log-likelihood function:

$$max_\theta \frac{1}{N} \sum_{n=1}^{N} log\ p(y_n|x_n)$$

Once the model is trained, it can be used to generate a target sequence given an input sequence. Though Seq2seq models were originally designed based on RNNs, other variants have emerged [169, 44]; among the more modern ones, a recent work introduces a sequence learning approach based on convolutional neural networks, which has shown to outperform many RNN-based models in the task of `NL-to-SPARQL` [197].

#### 2.3.1.1. Convolutional Sequence to Sequence Model

A Seq2seq model based completely on convolutional neural networks (CNNs) is proposed by Gehring et al. [61], called the Convolutional Sequence-to-Sequence model (ConvS2S). For this subsection we will assume a basic understanding of CNNs, where more details about this topic can be found in Appendix A.

Since CNNs do not receive the input as a sequence like RNNs do, a **position embedding** is proposed. First, the input elements $x = (x_1, \ldots, x_m)$ are embedded in distributional space as $w = (w_1, \ldots, w_m)$, where $w_j$ is a column in an embedding matrix $D$. These embeddings are combined with an absolute position vector $p = (p_1, \ldots, p_m)$, which indicates the position of the word in the sequence, in order to establish a sense of order in the input. From this combination the input element representation $\mathrm{e} = (w_1 + p_1, \ldots, w_m + p_m)$ is obtained. The output elements generated by the decoder are built using a similar representation.

To compute intermediate states, a simple **convolutional block structure** is used for both encoder and decoder, where such blocks are also referred to as *layers*. These intermediate states are based on a fixed number of input elements whose output for the l-th block are denoted as $z^l = (z_1^l, \ldots, z_m^l)$ for the encoder network and $h^l = (h_1^l, \ldots, h_m^l)$ for the decoder network. Each block contains a one dimensional convolution followed by a non-linearity.

Each convolution kernel is parameterized as $(W, b_w)$ and takes as input $X$, which is a concatenation of $k$ input elements embedded in d dimensions, and maps them to a single output element $Y$. Subsequent blocks operate over the $k$ output elements of the previous block. The gated linear unit (GLU) [41] was chosen as the non-linearity to apply over the output of the convolution $Y$. This gating mechanism permits controlling which input values of the current context are relevant. Aside from that, to enable deep convolutional networks, residual connections are added from the input of each convolution to the output of the block [82].

The input of the encoder network is padded to match the output length of the convolutional blocks for each block. The padding is done by adding $k - 1$ zero vectors on both the left and right side of the input to then remove k elements from the end of the convolution output. The same padding cannot be done for the decoder network since no future information is known beforehand [179].

A linear mapping is added for projecting between the embedding size and the convolution outputs. This mapping is applied to $w$ when feeding embeddings to the encoder network, to the encoder output $z_j^u$, to the final block of the decoder just before the softmax $h^L$, and to all decoder blocks $h^l$ before computing the attention score, which is explained later.

Lastly, a distribution is calculated over the $T$ possible next target elements $y_{\mathrm{i}+1}$ by transforming the top decoder output $h_{\mathrm{i}}^L$ via a linear layer with weights $W_o$ and bias $b_o$, as follows:

$$p(y_{\mathrm{i}+1}|y_1, \ldots, y_{\mathrm{i}}, x) = softmax(W_o h_{\mathrm{i}}^L + b_o)$$

Besides the convolutional block structures, a separate **attention mechanism** is implemented for each decoder layer. Attention allows the model to focus on the relevant parts of the sentence for each time step. The entire process is illustrated in Figure 2.7. The calculation of the attention starts in the bottom left part of Figure 2.7 with a combination between the current decoder state $h_{\mathrm{i}}^l$ and the embedding of the previous target element $g_{\mathrm{i}}$, which is defined as:

$$\mathrm{d}_{\mathrm{i}}^l = W_{\mathrm{d}}^l h_{\mathrm{i}}^l + b_{\mathrm{d}}^l + g_{\mathrm{i}}$$

Then looking at the center part of Figure 2.7, for a decoder block $l$, the attention $a_{\mathrm{i}j}^l$

of state i and source element $j$ is computed as a dot-product between the decoder state summary $\mathrm{d}_\mathrm{i}^l$ and each output $z_j^u$ of the last encoder block $u$:

$$a_{\mathrm{i}j}^l = \frac{exp(\mathrm{d}_\mathrm{i}^l \cdot z_j^u)}{\sum_{\mathrm{i}=1}^m exp(\mathrm{d}_\mathrm{i}^l \cdot z_j^u)}$$

Subsequently, the conditional input $c_\mathrm{i}^l$ to the current decoder block is a weighted sum of the encoder outputs as well as the input element embeddings $\mathrm{e}_j = w_j + p_j$ which corresponds to the center right part of Figure 2.7, and is defined as:

$$c_\mathrm{i}^l = \sum_{j=1}^m a_{\mathrm{i}j}^l (z_j^u + \mathrm{e}_j)$$

Finally, the conditional input $c_\mathrm{i}^l$ is added to the output of the corresponding decoder layer $h_\mathrm{i}^l$, as seen in the bottom right part of Figure 2.7. Compared with the classical single step attention, this proposal is named a *multi-step attention* mechanism since each step takes into account the attention history of the previous time steps, based on how conditional inputs are computed. Thus, information does not struggle to survive several steps as happens with recurrent networks.

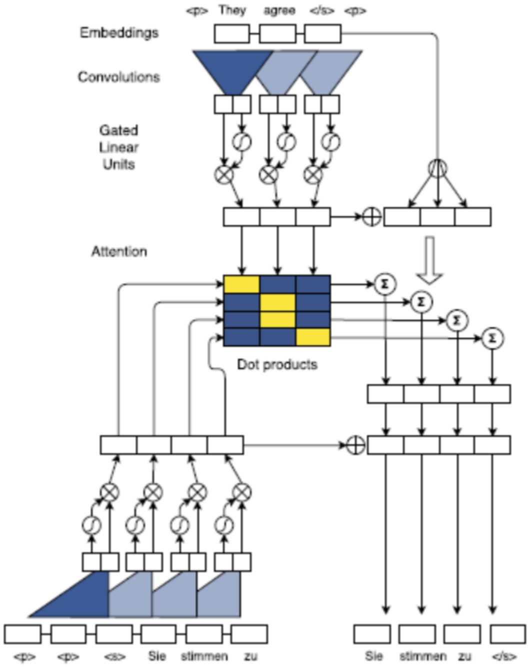


Figure 2.7: Convolutional block structure with a multi-step attention mechanism [61].

Some **normalization strategies** are applied to stabilize the learning process. First, the input and output of a residual block are summed with $\sqrt{0{,}5}$ to halve the variance of the sum. Second, the conditional input $c_i^l$ are scaled by $m\sqrt{1/m}$ to counteract any change in variance. Lastly, for convolutional decoders with multiple attention, the gradients for the encoder blocks were scaled by the number of attention mechanisms used, excluding source word embeddings.

Besides scaling, a **weight initialization** is also done to keep variance retained. Embeddings are initialized from a normal distribution with mean 0 and standard deviation 0.1. Weights from layers whose output is not directly fed to a GLU are initialized from $\mathcal{N}(0, \sqrt{1/n_l})$, where $n_l$ is the number of input connections to each neuron. This helps to maintain a variance of the input with a normalized distribution. Layers followed by a GLU are initialized from $\mathcal{N}(0, \sqrt{4/n_l})$, which is a weight initialization scheme based on works by He et al. [82] and Glorot & Bengio [68]. Biases are uniformly set to zero. Lastly, dropout is applied to the input on some layers with a probability of p of being retained and so scaled by $1/p$, or setting them to zero otherwise [169].

### 2.3.2. Natural Language to SPARQL

Though most works have focused on translating natural language to `SQL` queries [30, 204], recent work has also addressed the translation of natural language to `SPARQL`. In particular, Neural Machine Translation (NMT), which involves Machine Translation systems based on Neural Networks, has been used to develop systems that translate questions to `SPARQL` queries. An evaluation of eight different NMT models was performed by Yin et al. [197]. The NMT models included in this work were based on the aforementioned encoder-decoder Seq2seq architecture. Among these models, six of them are based on recurrent neural networks (RNN), while the remaining two are based on the ConvS2S model [61] and the Transformer model [180] respectively.

The systems based on RNNs include a baseline and many variants of the same LSTM architecture and different types of attention mechanisms. As a baseline, a Neural SPARQL Machine (NSpM) [162, 163] is used, which consists of a 2-layer LSTM model with no attention mechanisms. Then, two variants of the NSpM are used with two different types of attention: global attention and local attention. Global attention uses the entire input sentence to calculate an attention vector, which complements the context vector output by the encoder [11]. On the other hand, local attention only uses a fixed size window around every word of the sequence to calculate a scoped attention vector per word [115]. Another model is based on a proposal from Luong et al. [115], which consists of a 4-layer LSTM model with local attention. Lastly, the Google Machine Translation (GNMT) architecture [190] is included with two different variants: a 4-layer LSTM and a 8-layer LSTM, both with local attention. The only difference between Luong's model and the GNMT is that the second model includes residual connections from the third layer and uses a bidirectional LSTM in the first layer of the encoder.

#### 2.3.2.1. Neural Machine Translation

Though the architecture of each model varies, the encoding of SPARQL queries used in all approaches is the same as that proposed by Soru et al. [162]. Their encoding suggests that URIs are abbreviated using their prefixes followed by an underscore; brackets and dots are replaced by their verbal description, and SPARQL keywords are lower-cased. For example, a query over Wikidata is shown in Listing 2.5, when after an encoding conversion, it is converted to an encoded query as seen in Listing 2.6.

```
PREFIX wd: <http://wikidata.org/wiki/>
PREFIX wdt: <http://wikidata.org/prop/direct/>

SELECT ?sbj
WHERE {
    ?sbj wdt:P19 wd:Q201007 .
    ?sbj wdt:P166 wd:Q37922 .
}
```

Listing 2.5: SPARQL query before encoding.

After training, the system will output an encoded form that can be transformed back to the original SPARQL representation.

```
select var_sbj where brack_open var_sbj wdt_p19 wd_q201007 sep_dot var_sbj wdt_p166
    ↪ wd_q37922 sep_dot brack_close
```

Listing 2.6: SPARQL query after encoding (note it excludes the PREFIX clauses).

The evaluation metrics typically used to compare these systems are string-matching Accuracy, BLEU score and Perplexity. The string-matching Accuracy is used to measure the amount of exact matches delivered by each system. The global Accuracy is then the percentage of cases that are syntactically equal to the expected answer. This metric is particularly useful when measured over a dataset based on SPARQL templates, giving insights into whether or not the expected SPARQL form is being correctly generated.

The Bilingual Evaluation Understudy (BLEU) score is used to measure how similar two sentences are by using a geometric average of modified n-gram Precision [139], which is represented by the following formula:

$$BLEU = BP \cdot exp(\sum_{n=1}^{N} w_n \, log \, p_n)$$

The modified Precision $p_n$ for each candidate counts the number of times an n-gram occurs in a reference translation(s), takes the maximum count of each n-gram among the reference(s), and then clips the count of each n-gram in the candidate translation to the maximum count. To avoid short candidate translations getting higher scores than desired, a brevity penalty (BP) is applied which is set to 1 if the candidate length $c$ is larger than the maximal reference length $r$ or set to $exp(1 - r/c)$ otherwise. The $w_n$ represents weights for each modified Precision. By default $w_n = \frac{1}{N}$ and $N = 4$. In this case, the BLEU score goes from 0 to 100, where a score closer to 100 means the model is performing better. Note that BLEU does not account for word order.

**Perplexity** is used to understand the model's intrinsic behavior based on a cross entropy H(p,q) which is defined as follows:

$$H(p,q) = -\sum_{x} p(x)\ log\ q(x)$$

where $p$ represents the target probability distribution and q is the estimated probability distribution. Their similarity is defined by all possible values $x$ in the distribution. In this case, $p$ is the one-hot encoding vector of the target vocabulary and q is deduced from the result of the output softmax layer. Then the Perplexity is defined as the exponentiation of the cross entropy:

$$perplexity(p,q) = 2^{H(p,q)}$$

According to the experiments conducted by Ying et al. [197], the model that performs best was the ConvS2S model. Although many datasets were used in their experiments, we are only interested in `LC-QuAD 1` [47] and `DBNQA` [81], two datasets that represent opposite traits of a Question Answering Dataset: the `LC-QuAD 1` dataset contains few questions (5000) but with high complexity and high variance of its questions while the `DBNQA` dataset contains a huge volume of questions (nearly $900,000$) but it lacks variety in its questions. We explain with more details what we understand by a "good dataset" in the *Question Answering* section 2.4. The results of the three best models among the eight selected over the mentioned datasets are shown in Table 2.2.

| | **Perplexity** | | | | **BLEU Score** | | | | **Accuracy** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **LC-QuAD 1** | | **DBNQA** | | **LC-QuAD 1** | | **DBNQA** | | **LC-QuAD 1** | | **DBNQA** | |
| **Models** | **Train** | **Valid** | **Train** | **Valid** | **Valid** | **Test** | **Valid** | **Test** | **Valid** | **Test** | **Valid** | **Test** |
| LSTM_Luong | 1.12 | 4.92 | 1.90 | 2.15 | 52.43 | 51.06 | 77.64 | 77.67 | 0 | 0 | 34 | 34 |
| ConvS2S | 1.14 | 3.25 | 1.12 | 1.25 | 61.89 | 59.54 | 96.05 | 96.07 | 8 | 8 | 85 | 85 |
| Transformer | 1.16 | 3.15 | 2.21 | 3.34 | 58.99 | 57.43 | 68.68 | 68.82 | 7 | 4 | 3 | 3 |

Table 2.2: Performance comparison of three models included in Yin et al.[197].

Based on the Perplexity results, there is a serious overfit for all models over the `LC-QuAD 1` dataset which Yin et al. attributes to the small size of the dataset and its complex questions. No evident overfit is spotted over `DBNQA`, and the ConvS2S shows better results which is reflected in the other results as well. The BLEU score results reflect that ConS2S again outperformed all other models and shows a correlation between Perplexity and BLEU, especially when looking at DBNQA results. Finally, Accuracy results clearly show that RNN based models and the Transformer do not perform positively in any case when compared to the ConvS2S model. However, the results over the LC-QuAD 1 dataset shows there is still a challenge regarding the handling of more complex questions among all NMT models.

## 2.4. Question Answering over Knowledge Graphs

Question Answering systems respond to the need to access information on the Web without detailed knowledge of Semantic Web technologies such as how data is structured (RDF) or how to access data (SPARQL). In particular, Question Answering systems (QASs) provide end-users with an intuitive and easy-to-use user interface, which hides the complexity behind the Semantic Web standards [177]. These systems differ from traditional search engines such as Google in the final objective: search engines only return documents in which the answer can be potentially found, whereas a Question Answering system aims to return precise answers [113].

When we mention the task of Question Answering over Knowledge Graphs (KGQA), we refer to receiving a natural language question and returning an answer retrieved from one or more Knowledge Graphs (e.g. the first man to walk on the moon is Neil Amstrong[13]). Though there is work that includes a wider context along with the question, such as hybrid questions or chain of questions, we focus only on the problem of responding to an individual question without further context besides the question itself.

Another consideration for defining the scope of Question Answering is the question and answer type a system aims to respond to/with. Among the types of questions for which a Question Answering system usually provides an answer are:

- **Definition question**, refers to a definition of a subject or object (e.g. *"Who was Violeta Parra?"*).
- **Factoid questions**, which are related to facts. This type of questions includes three different variants:
  - **Predicative questions** that refer to a specific object related to a predicate such as who, what, where, how (e.g. *"Who was the first man in space?"*, *"What is the highest mountain in Chile?"*).
  - **List questions** that refer to all the answers that fulfill the fact being asked (e.g. *"Give me all the countries in America"*).
  - **Boolean questions** that refer to whether the fact being questioned is true or

[13] https://www.wikidata.org/wiki/Q1615 in Wikidata.

not (e.g. *"Was Gabriela Mistral a poet?"*).

In the context of Linked Data, a Question Answering system is limited to the information the available knowledge graphs can represent. For instance, questions not based on specific facts such as process questions (e.g. *"How do I make a lemon pie?"*) or opinion questions (e.g. *"What do most Chileans think of global warming?"*) cannot be answered.

Questions can also be classified depending on the answer type expected. One example is the work of Li et al. [108], which classifies questions given five high-level categories: **entities** (e.g. event, color, animal, plant), **descriptions** (e.g. definition, manner, reason), **humans** (e.g. individual, group), locations (e.g. city, country, mountain) and numbers (e.g. count, date, distance, size). Another way is to classify questions according to their **focus** and **topic**, representing what the question is about. For instance, the question *"What is the height of Aconcagua mountain?"* focuses on the property height in the topic of geography while the question *"What is the best height-increasing drug?"* focuses on the same property, but a different topic, which is medicine.

According to Fu et al. [59], current systems do not struggle with answering simple questions, i.e. questions that only require one subject-predicate-object triple fact, but complex questions are still a challenge for KGQA. A complex question usually is referred to as three types: questions under specific conditions (e.g. *"Who was the president of Chile during the Great Recession of 2008?"*), questions with more intentions (e.g. *"Give the names of the Chilean national football team and the number of goals they have scored"*), and questions requiring constraint inference (e.g. *"What is the most expensive movie starring Pedro Pascal?"*).

### 2.4.1. KGQA approaches

Though there is no standard approach for KGQA, most techniques or methods proposed follow a similar four-stage pipeline [43]. First, a question analysis stage includes techniques that extract information from the current question using purely syntactic features. Then, the phrase mapping stage defines techniques that identify KG resources for each named entity and its dependencies. Next, the disambiguation stage encapsulates techniques that rank and determine for each named entity the most relevant resources according to the context of the question. Lastly, the query construction stage considers techniques used to construct the `SPARQL` query that should retrieve the final answer. In summary, most Question Answering systems are a combination of these four steps and vary with respect to which techniques are used to address each phase of the Question Answering process.

On the other hand, KGQA approaches can be classified according to which techniques their models are based on. The traditional KGQA models rely on predefined templates and manuals to parse questions [43]. However, these traditional approaches require a certain level of knowledge of linguistics, and tend to be difficult to scale. Therefore, recent work has focused more on two kinds of approaches: methods based on Information Retrieval methods and other ones based on Neural Semantic Parsing.

#### 2.4.1.1. Information Retrieval-based methods

Information Retrieval (IR) based models reduce the QA task to binary classification or sorting over candidate answers. An IR-based method usually begins by identifying relevant entities mentioned in the question. Then, it extracts topic-entity-centric subgraphs where all nodes are considered candidate answers. Next, candidates are scored using features that provide semantic relevance and help to select the final answers. Depending on how the feature representations are built, IR-based methods can be divided into those based on feature engineering and those based on representation learning.

Methods based on feature engineering rely on manually defined and extracted features. For example, Yao et al. [194] extract four types of features based on the question's syntactic information, which includes question words, question focus words, topic words, and central verbs. These features are used in a classification model to determine the final answer. However, building these features is time-consuming and is not able to capture the entire semantic information of questions.

Methods based on representation learning convert questions and candidate answers into vector representations and reduce KGQA to a semantic matching computation between representations of questions and their candidate answers. Some methods incorporate external knowledge to complement the representation information and address the incompleteness that the KG might have [192, 168, 175]. Some other methods incorporate a multi-hop reasoning process to handle more complex questions [167, 129, 148].

Overall, IR-based models get rid of manually defined templates and rules, and can follow an end-to-end training. However, these methods lack model interpretability and are not able to handle complex questions that require constraint inference.

#### 2.4.1.2. Neural Semantic Parsing-based methods

In this context, methods based on Semantic Parsing aim to convert natural languages into executable query languages. Methods based on Neural Semantic Parsing (NSP) rely on Neural Networks to enhance the parsing capacity and scalability, instead of relying on predefined templates or rules. Thus, these models aim to map unstructured questions to intermediate logical forms which are later converted to structured queries.

One approach is to use query graphs, which are graphs that encoded questions, have strong representation ability and share topological commonalities with knowledge graphs. One example is GraphParser, proposed by Reddy et al. [152] which frames the Semantic Parsing problem as a Graph Matching problem. Derived from GraphParser, the Staged Query Graph Generation [195] (STAGG) and the Multiple Constraint Query Graph [13]

(MultiCG) were proposed. While STAGG proposed a different construction of the query graph based on a restricted subset of lambda-calculus in the graph representation, MultiCG extends STAGG to include more constraint types and operators in order to cover more complex questions. Since all these proposals rely on Entity Linking tools to initialize the query graph construction, Yu et al. [200] assumes a poor performance of such tools and proposes a Hierarchical Residual BiLSTM with the goal of improving entity recognition Accuracy.

On the other hand, other work proposes to use encoder-decoder models to reduce the Semantic Parsing task into a Sequence-to-Sequence problem. One example is the Seq-to-Tree model proposed by Dong et al. [44] that uses a hierarchical tree-structured decoder to capture the structure of logical forms. Then, in order to take more advantage of the syntactic information of the input question, the Graph-to-Seq model was proposed by Xu et al. [193] to encode the question as a syntactic graph. A last example is the Neural Symbolic Machine proposed by Liang et al. [110], which proposed a lighter supervision approach by training a Seq2seq model with reinforcement learning. Nevertheless, none of the examples mentioned above have been evaluated in the context of KGQA.

In general, the results of NSP-based models tend to be slightly better than the results from IR-based models. Nevertheless, it is still challenging to train a Neural Semantic Parser due to the lack of training data.

### 2.4.2. Main Challenges

The KGQA task is still an open problem, where many challenges have partially been addressed. Some of these challenges are related to the question being asked and others to the answers that can be returned.

There are different ways to express the same question, as there are different ways to represent information within Knowledge Graphs. The **lexical** gap is the difference between the vocabulary used in a question and how information is expressed in the Knowledge Graph [77]. The bigger this gap is, the more difficult it is for QA systems to locate the correct resources for each named entity identified.

Some work has tried to address the lexical gap, where most techniques are based on string normalization, pattern matching or entailment. For example, string normalization methods, such as converting to lower case or stemming (e.g. convert *"writing"*, to its base form *"write"*) help to reduce the search space when mapping entities. Another example are pattern libraries, such as PATTY [133] or BOA [62], that support pattern matching from a phrase to a resource. Nevertheless, most techniques proposed are based on manually defined rules, which is hard to scale or to transfer to other Knowledge Graphs.

Another challenge related with input expressivity is the **ambiguity** bound to each question, in other words, questions with different semantic meaning but with similar syntactic structure. There are two main types of ambiguities: homonymy, when different concepts are represented by a string with the same spelling (e.g. the word *"right"* can refer to *"correct"* or *"direction opposite to left"*), and polysemy, when one string can represent different but related concepts (e.g. *"newspaper"* can refer to a *"printed publication"* or to a *"media company"*).

Among the approaches used to address ambiguity are corpus-based methods or resource-based methods. Usually the corpus-based methods use statistical models from unstructured text corpora [160, 159]. On the other hand, resource-based methods take advantage of the RDF properties of candidate resources. A score is calculated for each entity following the assumption that a better score applies a higher probability of a resource to be chosen. Some examples are RVT, which uses Hidden Markov Models [64]; CASIA, which relies on Markov Logic Networks [161]; or Treo [57, 58], which uses Wikipedia-based semantic relatedness.

Unlike the lexical gap, which affects the Recall of a QA system, ambiguity negatively affects its Precision. While some methods aim to reduce the lexical gap, these same methods can make it difficult to address ambiguity. It is customary that in the disambiguation stage the systems try to balance the effects of both issues.

From challenges that are related to the query construction, the needs of **complex operators** affect the capability of a QA system to respond to more complex questions. The difficulty to handle a question increases when several facts have to be identified. Some examples that have tried to addresses these issue are YAGO-QA [3], which tries to address nested questions; PYTHIA [176], which can answer question involving quantifiers, comparatives, superlatives and others; and IBM Watson [67], which can respond to indirect questions and multiple sentences.

As mentioned before, there are certain types of questions that current QA systems struggle to handle. One example is that of **procedural questions** (i.e. describe a procedure), which no QA system has been able to solve. Another example relates to **temporal questions** which refer to questions that require inferring temporal relations between events, though some works have tried to address temporal questions [6, 53, 127]. A last example relates to **spatial questions** that include questions referring to locations. The capability of answering spatial questions depends on how the schema of the knowledge represents locations (e.g. latitude and longitude), and how QA systems can use that information to enrich semantic data [199, 205].

The use of **templates** is thus more common to construct SPARQL queries that include complex operators such as aggregation or filter functions. These SPARQL Query Templates can be either manually or automatically created. One template-driven approach is Casia [161] that uses the question type, named entities and POS tagging techniques to generate graph pattern templates from which a SPARQL query is built. Other approaches use manually created templates combined with machine learning methods [1].

### 2.4.3. Benchmark & Datasets

In order to measure a Question Answering system performance, the most common parameters used across all benchmarks are **Precision**, **Recall** and **F1-score**. These three metrics usually are based on a gold standard set per question, which are the entities or values expected to be returned by the QA system. Given a question $q$, the formulas for Recall, Precision and F1-score are represented in the following formulas:

$$\begin{aligned} Recall(q) &= \frac{\text{number of correct system answers for q}}{\text{number of gold standard answers for q}} \\ Precision(q) &= \frac{\text{number of correct system answers for q}}{\text{number of system answers for q}} \\ F1(q) &= \frac{2 * Precision(q) \cdot Recall(q)}{Precision(q) + Recall(q)} \end{aligned} \tag{2.3}$$

As an example, given the question *"Which are the primary colors?"*, the gold standard answer would be {`red`, `green`, `blue`}. If a system answer is {`brown`, `green`}, the Precision would be 0.5 since `green` is the only color part of the correct answers among the two colors returned, while the Recall would be 0.33 because the only color correctly returned among the golden standard set is `green`.

Similar to Entity Linking system evaluations, the global Precision or global Recall can be reported in two ways: as a micro average or a macro average. The F1-score is also used to combine Precision and Recall into one measure.

In recent years, many datasets have been developed to include complex questions. In particular, we will briefly describe the datasets used in this work whose questions often require complex query construction and provide the expected query and result.

#### 2.4.3.1. QALD

The series of QALD datasets are part of the Question Answering over Linked Data (QALD) challenges, which aim to provide an up-to-date benchmark for measuring performance and comparing Question Answering systems [112]. The QALD challenge is divided into multiple tasks that invite QA systems to address different challenges of Question Answering: multilinguality, hybrid questions, large-scale question answering and adaptability to other data sources. Though most versions of QALD contain questions that can be only answered over DBpedia, one of the versions of QALD did contain questions over Wikidata for the task of adaptability.

That being said, the $7^{th}$ version of QALD (`QALD-7`) [178] includes a dataset of 150 questions over Wikidata, divided into 100 questions for training and 50 for testing. The questions comes from a real-world question and query logs, where each question is manually annotated and includes a manually specified SPARQL query. Regarding the complexity of the questions, about 38 % are considered complex, including questions with counts, superlatives, comparatives, and temporal aggregations.

Despite the good quality of the questions of QALD-7 and its proximity to real-world questions, its limited size means that it is insufficient for training NSP-based models, which require a considerable amount of annotations to undergo a successful learning process

### 2.4.3.2. LC-QuAD 1

The Large-Scale Complex Question Answering Dataset (LC-QuAD 1) is a dataset based on DBpedia [174], where 82 % of its questions are considered complex. Differently from QALD, the SPARQL queries are built using a relatively small number of predefined templates, thus generating a large high-quality dataset with low domain-expert intervention. This results in a dataset that contains a total of 5000 questions with their corresponding queries.

Their dataset construction pipeline follows a different proposal to the standard one, where instead of writing the natural language question and then its logical form (aka the SPARQL query), the process is inverted. Thus the process starts by filling 38 hand-made SPARQL templates with seed entities and relations from a preferred predicate list to generate specific SPARQL queries. After that, a natural language question (NLQ) is deduced, for each generated SPARQL query, following a three-step process: a generation of a normalized NLQ by filling a Normalized Natural Question Template (NNQT) with the entity and relations names used to generate the current query, a verbalization of the normalized NLQ using crowdsourcing tools, and a final validation done by an independent reviewer.

The proposal of LC-QuAD 1 was a first step to allow NSP-based models to be applied to the field of Question Answering in the RDF/SPARQL setting, though most works [59] show that a larger number of examples are still required to train models properly.

### 2.4.3.3. DBNQA

The DBpedia Neural Question Answering (DBNQA) [81] dataset is thus far the largest dataset for Neural Question Answering over DBpedia, with 894,499 annotated pairs. The process to construct this dataset is similar to the one proposed for LC-QuAD 1, though its templates are extracted from the multilingual QALD-7 train dataset that includes 215 questions (not the same dataset as the Wikidata QALD-7 dataset mentioned before) and the same LC-QuAD 1dataset. This extraction process provides 5217 SPARQL templates. Another difference is the approach to generate natural language questions. This dataset originated from the purposes of training Neural SPARQL Machines, mentioned in the *Semantic Parsing* section 2.3.

The template extraction process for each case consisted of replacing the concrete entities with placeholders. This is the case for the original question, where resource labels are replaced, and the query, where entities are replaced. This process functioned differently for the two datasets used. For the QALD-7 train [178] dataset the resources were manually replaced, while for LC-QuAD 1 an automatic script was implemented taking advantage of the fact that the resources were marked in the questions. As an example, given the question *"What are the artists that are born in Stockholm?"* which generates the query in Listing 2.7, the template extraction process would return a question template *"What are the artists that were born in <A>?"*, replacing the resource Stockholm, and would generate the Query Template shown in Listing 2.8.

```
SELECT DISTINCT ?sbj WHERE {
    ?sbj dbp:placeOfBirth dbr:Stockholm .
    ?sbj rdf:type dbo:MusicalArtist .
}
```

Listing 2.7: SPARQL query for the question: *"What are the artists that are born in Stockholm?"*.

```
SELECT DISTINCT ?sbj WHERE {
    ?sbj dbp:placeOfBirth <A> .
    ?sbj rdf:type dbo:MusicalArtist .
}
```

Listing 2.8: Query Template for the question: *"What are the artists that are born in <A>?"*.

Then, the dataset is constructed following a similar approach to the one used for LC-QuAD 1. First, a set of entities is chosen per Query Template, where the entities selected are the ones that contain the relations contained in the query (e.g. if the property location is contained in the query, places would be selected accordingly). Then the selected entities are used to generate each SPARQL query with its corresponding NLQ.

Since the construction of each case does not include a paraphrasing stage, there is almost no variation of questions that comes from the same template, i.e. there was no syntactic difference for questions with the same purpose. This lack of variation negatively affects the generalisation capabilities of NSP-based models trained on this dataset [16].

#### 2.4.3.4. LC-QuAD 2

Taking into account all the advantages and disadvantages of the aforementioned datasets, a second version of the Large-Scale Complex Question Answering Dataset (LC-QuAD 2) was proposed [47]. In particular, this dataset provides around 30,000 questions over Wikidata which contains complex questions and high diversity among its questions. The dataset generation workflow combines semi-automatic question generation along with a crowdsourced-based paraphrasing phase.

SPARQL queries are generated given a set of entities, predicates and templates, and do not differ much compared to the first version of LC-QuAD. Nevertheless, the SPARQL templates used are different from the ones used before. Now the templates are based on one of the 10 types of question LC-QuAD 2 aims to address:

1. **Single fact**: use a single (S-P-O) triple query, e.g. *"Who is the screenwriter of Mr. Bean?"*.

2. **Single fact with type**: the fact is focused on the type of constraint, e.g. *"Billie Jean was on the tracklist of which studio album?"*.

3. **Muti-fact**: use two connected facts, e.g. *"What is the name of the sister city tied to Kansas City, which is located in the county of Seville Province?"*.

4. **Fact with qualifiers**: includes more informative facts stored in qualifier properties, e.g. *"What is the venue of Barack Obama's marriage ?"*.

5. **Two intentions**: consider questions with two intentions and also rely on qualifiers, e.g. *"When and where did Barack Obama get married to Michelle Obama?"*.

6. **Boolean**: ask whether a fact is true or false, including questions with number as an object, e.g. *"Did Breaking Bad have 5 seasons?"*.

7. **Count**: uses the COUNT keyword to perform a numeric count over a certain fact, e.g. *"What is the number of Siblings of Edward III of England ?"*.

8. **Ranking**: requires counting and sorting in order to rank entities regarding a certain property, e.g. *"What is the binary star which has the highest color index?"*.

9. **String Operation**: applies string operations at a work or character level, e.g. *"Give me all the Rock bands that start with the letter R?"*.

10. **Temporal aspect**: covers temporal properties where various time facts are included in qualifier properties, e.g. *"With whom did Barack Obama get married in 1992?"*.

Considering that some types of questions might have more than one SPARQL template variation, a total of 22 templates are used.

Then, a set of relevant entities and a predicate list is selected based on each SPARQL template, meaning that each question type includes different entities and properties (e.g. the property birthPlace might not be used for questions that require counting). For each case, a subgraph is built based on three factors: an entity, the SPARQL template, and one or more suitable predicates. From each subgraph a SPARQL query is generated. Next, each SPARQL query is transformed to a normalized natural language question, called a Question Template (QT), which is then verbalized and paraphrased in a three-step pipeline based on

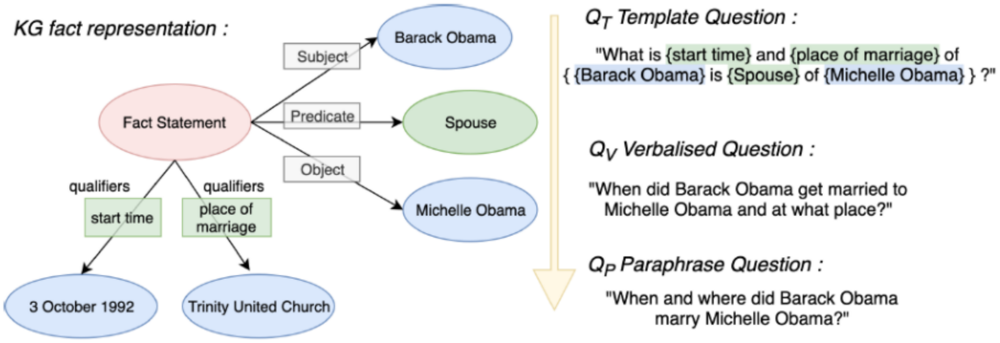

KG fact representation :
Fact Statement
Subject
Predicate
Object
Barack Obama
Spouse
Michelle Obama
qualifiers
qualifiers
start time
place of marriage
3 October 1992
Trinity United Church
$Q_T$ Template Question :
"What is {start time} and {place of marriage} of { {Barack Obama} is {Spouse} of {Michelle Obama} } ?"
$Q_V$ Verbalised Question :
"When did Barack Obama get married to Michelle Obama and at what place?"
$Q_P$ Paraphrase Question :
"When and where did Barack Obama marry Michelle Obama?"

Figure 2.8: Example of `LC-QuAD 2` question workflow generation [47].

human turkers from the Amazon Mechanical Turk tool (a crowdsourcing tool to perform simple tasks on a large scale).

The first step is to convert each QT into Verbalized Question (QV), where most of the grammatical errors and semantic inconsistencies of the QT are fixed. The second step is to paraphrase each QV, where the resulting Paraphrased Question (QP) should preserve the overall semantic meaning while changing its syntactic content and structure. An example of the entire generation of one case is illustrated in Figure 2.8. The last step is a human verification where a comparison is done between each QT with its corresponding QP in order to measure the quality of the final result in terms of semantic similarity.

After the entire generation process, a dataset is provided with around 52 % complex questions and significant variation in their question structure. There is, however, a small percentage of error derived from using a crowdsourcing tool [47], with questions losing part of their semantic intent or questions that are poorly verbalized/paraphrased, which should be taken into consideration when using this dataset to train NSP-based models.

# Chapter 3

# System Overview

In this chapter we describe the main components of the proposed Question Answering system. First we briefly provide a general overview of the proposed system structure, and then explain in more detail each of the modules implemented for this work.

## 3.1. Question Answering general overview

The Question Answering pipeline is divided into different stages, as seen in Figure 3.1, which do not differ much from the standard Question Answering pipeline described by Lopez et al. [43]. We assume that the target Knowledge Graph is Wikidata for the purposes of the example, but the architecture can be applied more generally to any Knowledge Graph. For the question *"Was Gabriela Mistral a poet?"*, the expected answer can be retrieved from the Wikidata query service by executing an ASK-type `SPARQL` query. In other cases a SELECT query may be required.

In order to build this `SPARQL` query, the Question Answering system starts with a Query Template Generation stage to generate `SPARQL` Query Template candidates with placeholders to be filled with entities in a later stage. These entities are first annotated in the Entity Linking stage, where relevant labels are identified (e.g., *"Gabriel Mistral"* or *"poet"*) and linked to their corresponding entity Knowledge Graph.

The process of filling the annotated entities on the placeholders of a Query Template is not a straightforward process. In the example provided in Figure 3.1, the entity of *"Gabriel Mistral"* (`Q80871`) should be the subject of a triple in a `SPARQL`query, but in other contexts this entity could be the object of a triple. Sometimes Entity Linking may recognize more entities than placeholders to be filled, thus adding another layer of complexity that an Entity Linking system cannot exclusively address. Therefore, the Slot Filling stage aims to identify which entities should be used to fill which placeholders of the Query Template candidates.

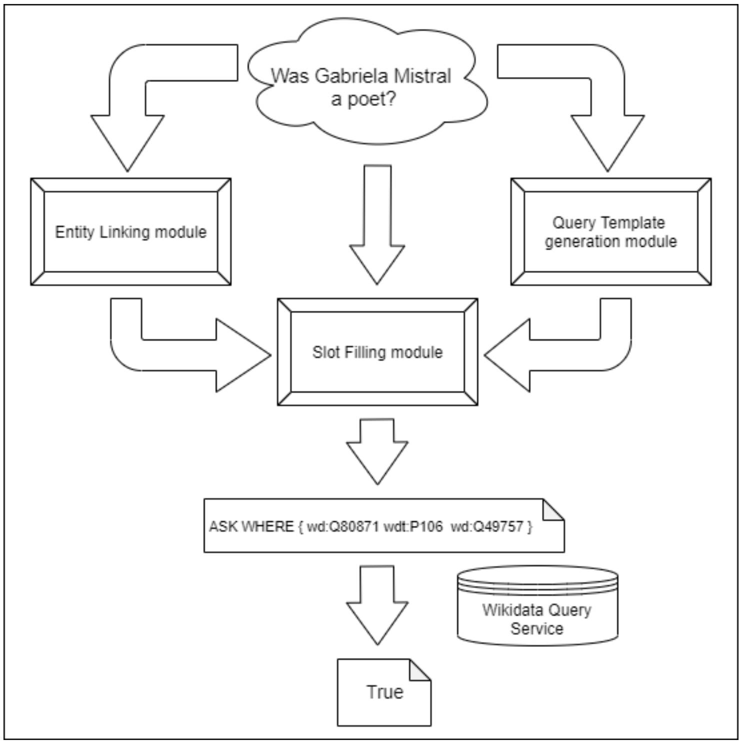

Was Gabriela Mistral a poet?
Entity Linking module
Query Template generation module
Slot Filling module
ASK WHERE { wd:Q80871 wdt:P106  wd:Q49757 }
Wikidata Query Service
True

Figure 3.1: Question Answering system pipeline.

The Question Answering pipeline described in the Figure 3.1 is implemented by building three modules that execute each one of the described stages.

## 3.2. Query Generation Module

In this section we describe the Query Generation Module, which is used by the Baseline system and adapted for the system proposed in this work. Given a Natural Language (NL) question, the goal is to generate its logical form in terms of `SPARQL` grammar, which can be either a complete `SPARQL` query representation (baseline) or an intermediate Query Template representation (our system).

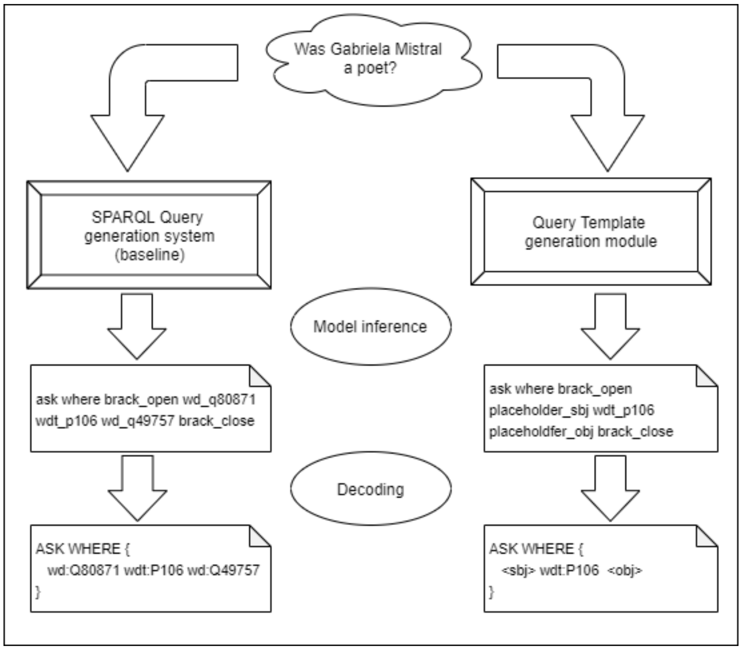
We now explain the main components that encapsulate the Query Generation module. First, we describe the Query Generation pipeline, which does not differ much between the generation of an entire `SPARQL` query or its Query Template. Then, we explain the Query Encoding used to deliver a normalized expression of the expected output for the model that performs the generation process. Finally, we describe in more detail how the model is trained and used to generate queries based on the FairSeq library for implementing Sequence to Sequence models in Python.

### 3.2.1. Query Generation pipeline

This module addresses two different tasks: translation from NL to SPARQL queries, and generation of Query Templates from NL.

On one hand, when we refer to SPARQL queries, we refer to complete queries with all the entities, properties, SPARQL keywords and operators necessary to be executable on any Wikidata endpoint. Previous works on this task are usually based on Sequence-to-Sequence models, where many have SQL as their target language [44, 30, 204]. Most recent works apply similar models targeting SPARQL [116, 162, 163].

On the other hand, Query Templates refer to intermediate representations of SPARQL queries, which instead of containing the necessary entities included in a complete SPARQL query, contain placeholders that will be later filled using the entities obtained in the Entity Linking stage. An example of both cases can be found in Figure 3.2, where the left side shows a SPARQL query, and the right side its Query Template.


Was Gabriela Mistral a poet?
SPARQL Query generation system (baseline)
Query Template generation module
Model inference
Decoding

```
ask where brack_open wd_q80871
wdt_p106 wd_q49757 brack_close
```

```
ask where brack_open
placeholder_sbj wdt_p106
placeholdfer_obj brack_close
```

```
ASK WHERE {
  wd:Q80871 wdt:P106 wd:Q49757
}
```

```
ASK WHERE {
  <sbj> wdt:P106  <obj>
}
```

Figure 3.2: Query Generation pipeline example.

In Figure 3.2, we can see the Query Generation pipeline and expected output for both tasks given the input NL question *"Was Gabriela Mistral a poet?"*. On the left, the SPARQL query pipeline outputs an ASK-type SPARQL question with two entities involved. On the other side, a Query Template is returned, which looks like an ASK-type SPARQL query but it replaces the entities for two placeholders indicating that there is a subject entity and an object entity expected in the first triple of the Query Clause. The idea of the Query Template is that these placeholders will later be filled by the Slot Filling module.

As we mentioned before, the Query Generation pipeline does not change whether the model is generating an entire SPARQL query or just an intermediate Query Template, and there are two reasons that explain this. First, both tasks can be seen as translating NL to a meaning representation, where such representations only vary on the information they are displaying. While generating a SPARQL query includes the entities involved in the query, the generation of a Query Template replaces those entities for placeholders indicating that in those places an entity or a plain value will be needed, without specifying which ones. The second reason is that both models rely on the same data: a set of pairs of NL questions and SPARQL queries. The only difference is that the output data used for the Query Template model is adapted to fit the Query Template generation task, where entities and plain values (e.g. numbers or strings values) are replaced by a placeholder that keeps a certain degree of information for the replaced value (e.g. placeholder type, query triple position, etc). This dataset adaptation process is explained in more detail in the *Experimental Design* chapter 4.

Aside from the input and output process, there are some intermediate steps performed over the input and output data. In the case of the input NL questions, a normalization process is conducted where the question string is lower-cased and non-relevant symbols are removed. On the output side, an encoding process is performed over the data used for training, and a decoding process is done for the query string output by the Query Generator system. The Query Encoding is done by tokenizing the SPARQL grammar, allowing for treating it as a target wordy language, following the idea of reducing this problem to a Machine Translation task used in previous works [35, 8, 197]. Then, each of the two Generator models shown in Figure 3.2 corresponds to a previously trained Fairseq model, which receives a normalized NL question and returns an encoded meaning representation that has to be later decoded to obtain the final SPARQL query or Query Template output.

### 3.2.2. Query Encoding

Following the same encoding approach used by Soru et al. [162], we encode the SPARQL queries into tokens that describe entities, keywords, symbols, and operations. One difference is that we are using a dataset that includes new query components such as numbers or string values; thus we propose novel tokens to encode those values. An example of this encoding can be seen in Listing 3.1. The main advantage of this encoding is that it allows the system to express chunks of commonly used patterns (e.g. order by asc/desc, or filter patterns) in a simpler way, thus reducing the complexity of the translation task.

```
ASK WHERE { wd:Q658 wdt:P1108 ?obj FILTER(?obj < 1.2) }

ask where brack_open wd_q658 wdt_p1108 var_obj filter attr_open var_obj math_lt 1
    ↪ _dot_2 attr_close brack_close
```

Listing 3.1: SPARQL query example with its encoded form.

Query Templates are encoded in a similar way, but instead of encoding entities, numbers or string values, placeholders are used to replace those values. Besides those components, there is no other difference in the encoding process. Listing 3.2 shows the Query Template of Listing 3.1 and its encoded form.

```
ASK WHERE { <sbj_1> wdt:P1108 ?obj FILTER(?obj < <num>) }

ask where brack_open placeholder_sbj_1 wdt_p1108 var_obj filter attr_open var_obj
    ↪ math_lt placeholder_num attr_close brack_close
```

Listing 3.2: `SPARQL` query example with its encoded form.

Note that instead of using letters for naming each placeholder as done by Ying et al. [197] (e.g. `A`, `B` or `C`) we use labels that provide more information about placeholders. For example, we identify placeholder type (subject, object, string value or numeric value) and position for entities placeholder (e.g. `sbj_1` corresponds to the subject of the first triple in the Query clause).

These placeholder labels are also used in the Slot Filling system mentioned later in order to identify which entities output by the Entity Linking system correspond to each one of the Query Template placeholders.

### 3.2.3. Fairseq Model

The Query Generation model is implemented using the *Facebook AI Research Sequence-to-Sequence Toolkit* (Fairseq), which implements various Seq2seq models based on Pytorch. Fairseq provides various off-the-shelf implementations and other settings to configure user experiments.

In particular, we use the Convolutional Sequence to Sequence model (ConvS2S) [61] implementation to build the Query Generator model. The decision to use the ConvS2S is based on the work done by Yin et al. [197], where eight Sequence-to-Sequence models were implemented and compared regarding the NL-to-SQL task. From this comparison, the ConvS2S model ended up having the best performance in terms of Perplexity, BLEU score, and string-match Accuracy. This model has to be trained and can be used later to generate an output depending on the given data. If we need to build a `SPARQL` Query generator, we have to feed the system with pairs of NL questions and `SPARQL` queries. In the same way, for a Query Template generator we instead use Query Templates as our output examples for training.

We now explain the main components of the ConvS2S architecture, the hyperparameters used with their values, and other training settings relevant to comprehend the training process. Note that the architecture used and its configuration is done following the work from Yin et al. [197].

The ConvS2S architecture components used are based on the best-performance settings for natural language NMT [197]. In particular, the most relevant hyperparameters are the followings:

- The number of **layers** used for the convolutional encoder and decoder is 15. From these layers, the first 9 layers use $3 \times 3$ kernels with 512 units, the next 4 layers use $3 \times 3$ kernels with 1024 units, and the final 2 layers use a $1 \times 1$ kernel with 2028 units.
- The **optimizer**, i.e. the learning algorithm used, is Stochastic Gradient Descent with minibatches. The default minibatch size is 64.
- The **learning rate** is set to a fixed value of 0.5, which is maintained during the entire training session.
- As a regularizer, **dropout** is applied with a fixed value of 0.2.

Besides those parameters, the Multi-step Attention mechanism is applied to some layers. The loss function used is the cross-entropy function mentioned in the *Theoretical Framework* chapter 2.3.1.

The training process is performed using Google Colaboratory, an online research tool that allows for writing and executing Python code and provides free computational resources, such as GPUs, to boost the training process. It is mainly focused on developing machine learning tasks. We also follow the same dataset split used by Yin et al. [197]: 80 % for training and 20 % for dev/test. One difference is we only use train and dev sets, since other datasets are proposed as test sets. We chose this strategy as some of the existing datasets we use contain related or paraphrased questions, where creating an independent test set ensures that test questions are not derived from, or variants of, training questions. Each model is trained for a maximum amount of 40 epochs, where after every epoch a checkpoint is saved. Then, the checkpoint that achieves the lowest loss value for the validation set is kept.

The best values obtained from the training process can be imported later to the same Fairseq model implementation to perform new evaluations. When decoding on the evaluation stage, beam search is applied with a beam width of 5. As a last note, the same normalization process done for the training data has to be done with every new NL question. On the other hand, to obtain the SPARQL query or the Query template it is always necessary to perform the decoded process over the Query Generator output.

## 3.3. Entity Linking Module

Two types of Entity Linking systems are considered for this work: Individual Entity Linking systems and Ensemble Entity Linking systems. First we discuss the pipeline that these systems follow.

### 3.3.1. Entity Linking pipeline

An example for the Entity Linking pipeline is shown in Figure 3.3, for the question *"Was Gabriela Mistral a poet?"*. We assume that the Entity Linking module is available through an API; the Entity Linking system first retrieves the annotations by querying the API service, which returns annotations along with extra information (e.g. scores, second ranking entities, etc). In the case that the Entity Linking system targets a different Knowledge Graph to the one over which questions are answered, in a second (optional) phase the entities are mapped from the former to the latter (we do not compute the mapping in this work but rather assume that a mapping is made available).

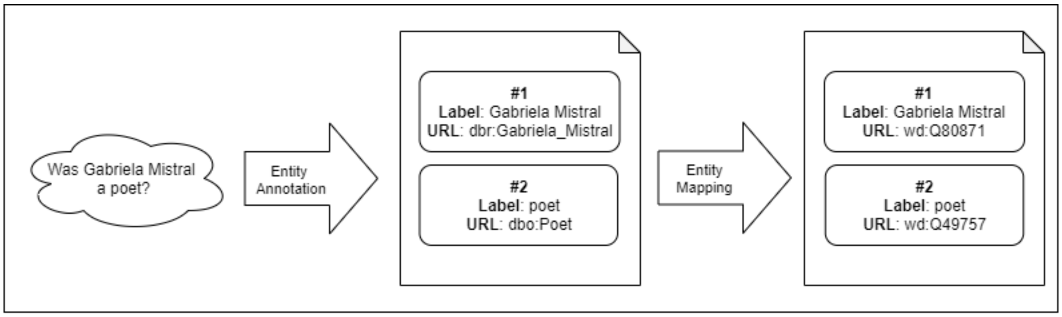


Figure 3.3: Entity Linking pipeline example.

In this example, we show for each spotted mention: the start and end position, the annotation label and its associated resource. Then, an entity mapping process is performed to convert the DBpedia resources (`dbr:Gabriela_Mistral` and `dbo:Poet`) into Wikidata resources (`Q80871` and `Q49757` respectively). As we mentioned, this mapping is optional and it depends on the target Knowledge Graph we are interested in, which in this case is Wikidata, and whether the system returns entities on that target Knowledge Graph.

The API service input parameters and output format is different for each web service, though the final output annotations from the Entity Linking module are always the same. An additional piece of information that is also returned are the scores each system gives to each annotation. Though these scores cannot be compared across systems, such scores are used to establish which entities are more relevant for each system.

The Entity Mapping process assumes an existing mapping between the entities of both Knowledge Graphs. In this particular case, we can take advantage of the fact that many DBpedia resources and Wikidata resources are both bound to Wikipedia articles via the `schema:about` property. This information can be found on Wikidata, thereby, given a set of DBpedia entities, a `SPARQL` query can be performed to retrieve their corresponding Wikipedia article and their related Wikidata entities, if any. An example of a `SPARQL` query to map DBpedia entities to Wikidata ones can be seen in Listing 3.3.

```
SELECT ?article ?wikidata ?dbpedia WHERE {
    ?article schema:about ?wikidata .
    BIND(IRI(CONCAT("https://en.wikipedia.org/wiki/", SUBSTR(STR(?dbpedia),29))) AS ?
        ↪ article)
    VALUES ?dbpedia { <http://dbpedia.org/resource/Gabriela_Mistral> <http://dbpedia.
        ↪ org/ontology/Poet> }
}
```

Listing 3.3: SPARQL query example to map DBpedia resources to Wikidata ones.

Following the same logic, Wikipedia articles can be mapped to their associated Wikidata resources. The opposite mapping from Wikidata resources to DBpedia ones can be easily performed as well. If any entity cannot be mapped from the results of the Entity Linking system to the target Knowledge Graph, it is discarded.

A final note about the entity mapping process is that it is performed over batches of entities, which means that a set of entities are mapped at the same time. This batch entity mapping is used to minimize the amount of queries done to the Wikidata query service. The final output values are returned in a JSON format.

### 3.3.2. Individual Entity Linking systems

We consider an Individual Entity Linking system as one of the Entity Linking systems mentioned in the Information Extraction chapter 2.2.2: DBpedia Spotlight [128], AIDA [198, 89], TAGME [52], and OpenTapioca [42]. These systems were selected because they provide public APIs that can be invoked over the Web. The annotation process is reduced to making a request on the corresponding API's web service. We will briefly describe some details of the implementation and mention some aspects to take into consideration for each one of the aforementioned systems.

The **DBpedia Spotlight** system [128] aims to annotate DBpedia entities, so it requires a mapping process to convert the output annotations into Wikidata entities. The input parameters for the API request are the query text, a confidence value, and a support value. The confidence parameter is a threshold for disambiguation, and the support parameter is used to filter resources with a small number of Wikipedia inlinks. On the output side, DBpedia Spotlight assigns two scores to each entity: a similarity score, which represents how similar are the mention and the entity, and a percentage of second rank, which is the ratio of similarity scores between the second and the first candidates of the corresponding mention (used to measure the level of ambiguity for the mention).

The **AIDA** system [198, 89] works over YAGO entities, though all YAGO entities map to a Wikipedia article. The annotations of this system can thereby be expressed as Wikipedia resources without the need for an extra query to the Wikidata query service. Nevertheless, the mapping process is done to map Wikipedia resources to Wikidata ones. Besides the text input parameter, the AIDA system does not require any other parameter. Each output annotation includes a disambiguation score which is the score assigned in the candidate ranking stage.

The **TAGME** system [52] makes annotations over Wikipedia. As per AIDA, it requires a mapping stage to convert Wikipedia resources into Wikidata ones. Aside from the query text input parameter, TAGME requires a token parameter for authentication, which can be retrieved from its API website. Each entity annotation has a *"rho"* score assigned, which is derived from the candidate ranking stage.

The **OpenTapioca** system [42] is the only one that works directly over Wikidata. It does not require any extra parameter besides the query text. In the output annotation, a logarithmic likelihood score is assigned to each annotation, which points out how prominent the entity is in the Wikidata Knowledge Graph.

As a general aspect, the GET requests of most of the systems' web services are done using the `request`[1] Python library, except for AIDA where `curl`[2] requests are done using the subprocess Python library. Then, the mapping queries performed on the Wikidata query service are done using the `SPARQLWrapper`[3] Python library.

### 3.3.3. Ensemble Entity Linking system

Besides each individual Entity Linking system, we propose an ensemble of Entity Linking systems in order to improve the performance in terms of recognizing all the entities in a given question. Since each individual system relies on different techniques or prioritizes different features, they can identify entities that are either very prominent or quite rare. In the context of Question Answering, if an Entity Linking system cannot identify all of the entities needed to build the required `SPARQL` query, the Question Answering system will fail to provide the right answer. The inability to identify all entities is more prone to happen when little context is provided in the given text, which is a common situation in Question Answering settings.

An ensemble Entity Linking system aims to make the most of all individual Entity Linking systems by combining their results together to achieve better Recall and Precision. Then, an ensemble system is given the annotations of each individual system as the system input and returns a final output annotation set. We present two variants to combine the individual annotations: Precision Priority system and Voting system. To better illustrate the variants proposed in this work, let us assume that each individual Entity Linking system returns the following annotations, for the question *"Does FC Barcelona have Juan Jose Ibarretxe as a chairperson?"*:

[1] `https://pypi.org/project/requests/`

[2] `https://curl.haxx.se/`

[3] `https://pypi.org/project/SPARQLWrapper/`

### AIDA

```
FC Barcelona - wd:Q7156
```

### OpenTapioca

```
Juan Jose Ibarretxe - wd:Q351738
```

### TAGME

```
Juan Jose Ibarretxe - wd:Q351738
chairperson - wd:Q140686
```

### DBpedia Spotlight

```
Does - wd:Q302057
FC Barcelona - wd:Q7156
Juan Jose Ibarretxe - wd:Q351738
chairperson - wd:Q140686
```

After the description of each variant, an example of the expected output annotations is given based on the results shown above.

In this example, the annotations expected to be recognized are the entities from *"FC Barcelona"* and *"Juan Jose Ibarretxe"*. Even though some systems recognized *"chairperson"* as an entity, the property *"chairperson"* (`wdt:P488`) is more likely to be used when building the `SPARQL` query for this question, so its entity is not expected to be identified. Though this ambiguity issue (whether something is a property or an entity) can complicate building the `SPARQL` query, it is not something that we address in the Entity Linking stage. Returning more entities than the number expected is still acceptable as long as the correct entities are included among the output annotations. However, we will see in the *Slot Filling module* section 3.4 that the more entities are passed from the Entity Linking stage, the more difficult it can be to find the correct entity-placeholder mapping. Thus, a threshold can be set to define the number of expected entities, which is commonly the number of slots contained in the predicted Query Template.

#### 3.3.3.1. Precision Priority system

The Precision Priority (PPrior) ensemble system establishes that some individual Entity Linking systems are more likely to return the expected answers, which in this case are the individual systems that tend to return fewer entities but with a high level of confidence, i.e. systems that prioritize Precision over Recall.

Then, the PPrior system assigns a priority to each individual system according to their performance on the datasets we are interested in (`LC-QuAD 2`, `DBNQA`, and `QALD-7`). In particular, systems with better Precision have higher priority. These results can be found in the *Results* chapter 5.2, which establishes the following priority: AIDA, OpenTapioca, TAGME, and DBpedia Spotlight. Next, the PPrior system retrieves the annotations of each individual system following the established priorities until the number of unique expected entities is reached. For example shown above, the PPrior system would rank each entity in the following way:

```
1. FC Barcelona - wd:Q7156
2. Juan Jose Ibarretxe - wd:Q351738
3. chairperson - wd:Q140686
4. Does - wd:Q302057
```

The first entity (`Q7156`) comes from the AIDA system, the second one (`Q351738`) from OpenTapioca, the third one (`Q140686`) from one of the annotations output by TAGME (since the other entity has already been considered), and the remaining one (`Q302057`) comes from DBpedia Spotlight. Therefore, if the number of expected entities is two, the entities of the mentions *"FC Barcelona"* and *"Juan Jose Ibarretxe"* would be output as final annotations.

One advantage of this variant is that it would perform more efficiently in an online setting since it only requires to request annotations from the individual systems until it fulfills its expected number of entities. A disadvantage to take into consideration is that the PPrior system relies on the idea that systems with higher priority make fewer mistakes (e.g. recognize fewer false positives); thus incorrect entities from higher priority systems would have more value than correct entities from smaller priority systems.

#### 3.3.3.2. Voting system

The Voting ensemble system establishes a voting scheme where annotations from each individual system are considered to be a vote that is used to rank entities. This is based on the idea that if an entity is widely recognized across Entity Linking systems, that entity is more likely to be a correct annotation. Note that votes go for the entity identifier and not for the mention. Therefore, if two individual systems recognize the same entity but differ on the mentions linked, both systems are voting for the recognized entity (e.g. the `Q351738` entity could have been assigned to *"FC Barcelona"* or only to *"Barcelona"*, but votes go to the `Q351738` entity).

Given that ties are possible, the entities that come from more precise systems would have priority (following the same reasoning as with the PPrior system). For example, the Voting system would rank each entity in the following way:

```
1. (3) Juan Jose Ibarretxe - wd:Q351738
2. (2) FC Barcelona - wd:Q7156
3. (2) chairperson - wd:Q140686
4. (1) Does - wd:Q302057
```

The most voted entity is `Q351738` with three votes, which comes from OpenTapioca, TAGME, and DBpedia Spotlight. Next, the entities `Q7156` and `Q140686` get the same amount of two votes. Since `Q7156` comes from a system with higher priority (AIDA), that entity ranks higher. The last entity `Q302057` only has one vote from the system with least priority (DBpedia Spotlight). Thereafter, if the number of expected entities is two, the entities of the mentions *"Juan Jose Ibarretxe"* and *"FC Barcelona"* would be output as final annotations.

The main advantage of the Voting system is that it can benefit from every system equally (all votes weigh the same regardless of their system's Precision). Thus, it seems reasonable to assume that the more systems are included in the voting process, the greater the chances are that the correct entities will be among the top voted ones. Some disadvantages are that with few systems the voting system does not differ much from the PPrior system, and its process is more expensive to execute since it always requires retrieving the annotations from all individual systems included.

#### 3.3.3.3. Other optimizations

In order to improve the performance of Ensemble Entity Linking systems, a couple of heuristic optimizations are proposed to achieve better performance. These improvements are included in the two variants mentioned before, and are optional features that can be skipped if no significant improvement is perceived.

First, some entities linked to stopwords (e.g. *"Does"* linked to `Q302057`) are filtered using an English stopwords list provided by the Natural Language Toolkit[4]. Before performing any join annotation process, the annotations that contain stop words are discarded. The idea behind this filter is to avoid entities from systems with low Precision negatively affecting the identification of correct entities. However, some entities may include stopwords in their names (e.g. the rock band *"The Who"*), where this approach may negatively affect the results for such entities.

Lastly, a tiebreak system is implemented for entities that are ranked in the same position and come from the same system. Since each individual system assigns a score to each annotation, such scores can be used to break the tie and decide which entities will be considered in the final output. For example, if only two entities are expected but the second and third are tied (e.g. in the Voting system let us assume the entities from *"Juan Jose Ibarretxe"* and *"chairperson"* comes from OpenTapioca and have two votes each), we look at which has a higher score according to the individual Entity Linking system these annotations come from

[4] `https://www.nltk.org/`

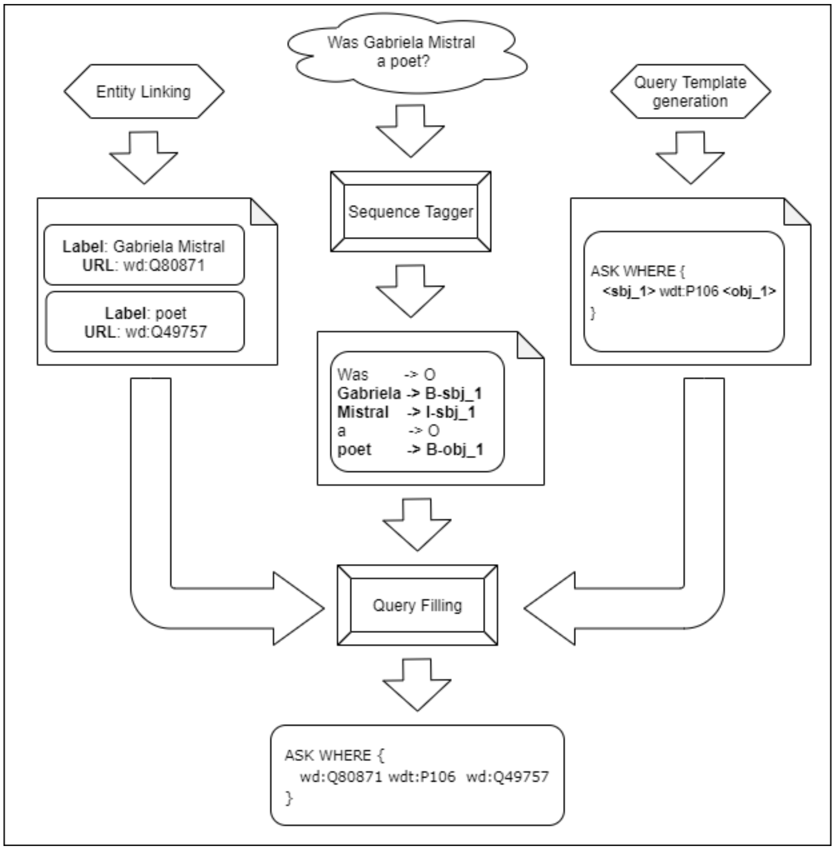
(i.e. which one has a higher `log_likelihood` score according to OpenTapioca).

# 3.4. Slot Filling Module

In this section we describe the main components of the Slot Filling module, which is divided into two stages: the sequence labelling stage done by a Sequence Tagger, and the Query Filling stage where the spotted entities are filled into the given Query Template. We explain with details how the Slot Filling pipeline is structured, and then describe how the Sequence Tagger and the Query Filling algorithm are implemented.

## 3.4.1. Slot Filling pipeline

In Figure 3.4, we can see an overview of the Slot Filling process. Differently from the other modules, the Slot Filling system not only receives the input NL question, but also the outputs from the Entity Linking system and the Query Template generator. The expected output is a `SPARQL` query which is based on the Query Template output by the Query Template generator, containing entities from the annotations returned by the Entity Linking system.



```
ASK WHERE {
  <sbj_1> wdt:P106 <obj_1>
}
```

```
Was       -> O
Gabriela  -> B-sbj_1
Mistral   -> I-sbj_1
a         -> O
poet      -> B-obj_1
```

```
ASK WHERE {
  wd:Q80871 wdt:P106  wd:Q49757
}
```

Figure 3.4: Slot Filling pipeline example.

Before the actual Slot Filling process, the input NL question is passed to a Sequence Tagger that identifies relevant named entities (or labels) and their corresponding placeholder tag. We understand a placeholder tag as a placeholder name in a Query Template. As seen in Figure 3.4, given the question *"Was Gabriela Mistral a poet?"*, the expected output for the Sequence Tagger is to identify *"Gabriela Mistral"* and *"poet"* as two relevant labels which are expected to have a Wikidata entity recognized by the Entity Linking system. For example, *"Gabriela Mistral"* is tagged as "`sbj_1`", meaning that the entity which corresponds to that label (`Q80871`) should be assigned to the placeholder `<sbj_1>` in the Query Template. The same happened with the label *"poet"*, which should be assigned to the placeholder `<obj_1>`. Note here that the goal is to infer label–placeholder annotations, whereas the Entity Linking system aims to produce label–entity annotations.

Next, the labels tagged with their corresponding placeholder are passed to the Query Filling stage, along with the Entity Linking annotations and the generated Query Template. A filling algorithm is proposed to perform the Query Filling process. The algorithm will return a `SPARQL` query only if the filling process is successful. As we will explain later, this filling process could fail due to the previous systems failing to recognize some entities, or due to a mismatch between the placeholders recognized in the tagging stage and the placeholder contained in the given Query Template.

### 3.4.2. Sequence Tagger model

The Sequence Tagger identifies which labels are more likely to have an entity assigned to a certain placeholder in the Query Template. To implement this model, we use the Flair[5] framework for Natural Language Processing. Flair contains many off-the-shelf Sequence Labelling and Name Entity Recognition models implemented using Pytorch. It also supports training a custom Sequence Labelling model if training data is provided.

We then adapt the same dataset used for generating the Query Template dataset to describe the expected output (more details in the *Experimental design* chapter 4.4.1). A Sequence Tagger model receives an NL question, and returns a sequence of tags where each tag is related to one word (or token) in the input question. These tags follow the BIO format, where each word is classified as the beginning of a tag, as the intermediate part of it, or as a non-tag token. For example, if given the question *"Was Gabriela Mistral a poet?"*, the expected output would be `(O, B-sbj_1, I-sbj_1, O, B-obj_1)`. From this output we can infer that *"Gabriela Mistral"* is identified as one tag (*"Gabriela"* is the beginning of the tag and *"Mistral"* is an intermediate part of it), and *"poet"* is identified as another tag. On the other hand, the words *"Was"* and *"a"* are identified as non-tag tokens. Note that the length of the tagger output and the number of tokens in the input question are always the same amount, so every token is assigned with a BIO tag.

In this work three **types of placeholders** are chosen for three types of values: entities,

[5] `https://github.com/flairNLP`

numbers, and string values. The entity placeholder identifies entities from the Knowledge Graph resource (e.g. `Q80871`). Each entity placeholder identifies two aspects of the entity: 1) its role in the query triple (subject or object), and its query triple position. For example, if an entity is tagged as "`sbj_1`", it means it is the subject of the first query triple. As their names may suggest, the number placeholder identifies numeric values, whereas the string value placeholder marks text relevant to the query (e.g. for filtering entities that contain a certain string label).

There are two main steps before training the Sequence Tagger model: preprocessing the dataset, and configuring the model architecture. The Sequence Labelling dataset has to be converted to a tag dictionary that fits the Flair model input. Each output value has to be expressed as a sequence of BIO tags. Whatever BIO tags are chosen determines the tags the Sequence Tagger will use. Next, the configuration of the model architecture is divided into embeddings and model settings. First, the embedding layer is set, where one or more types of embeddings can be joined into stacked embeddings. In this case, we use the GloVe embeddings along with contextual embeddings, also known as Flair embeddings [4]. Lastly, the model architecture used is the same standard architecture used by Akbik et al. [5], which is a BiLSTM [93] with the number of hidden layers set to 256, along with a CRF layer put on top of the final layer.

Finally, there are some training parameters and settings that are defined by default and others that can be tuned for a particular task. Some default aspects include the learning algorithm (mini-batch stochastic gradient descent) and the loss function (cross-entropy function). Some relevant training parameters that can be tuned are the learning rate, the minibatch size, and the number of epochs. The model is trained until it has a fixed number of epochs without an improvement in terms of loss value.

### 3.4.3. Slot Filling method

The placeholder tags for each named entity, along with the annotations from the Entity Linking system and the Query Template from the Query Template generator, need to be combined together to build the final `SPARQL` query output. The Slot Filling method decides which placeholder to replace with which entity, and is based on a novel filling algorithm divided into two filling stages. The first stage is a **Standard Filling** process that may not fill all the placeholders in the query, and the second stage is a **Force Filling** process that tries to complete the spots that the previous stage was not able to fill.

The first stage, the Standard Filling process, follows a straightforward process of comparing labels from the outputs of the Entity Linking system and the Sequence Tagger to identify entity-slot matches which are then inserted into the Query Template to form the `SPARQL` query. Then, the algorithm that describes the standard filling process is shown in Listing 3.4. As an example from the input shown in Figure 3.4, the algorithm would first compare the label *"Gabriela Mistral"* with all the labels from the entity annotations. Then, it would be found that the placeholder `sbj_1` should correspond to the entity `wd:Q80871` given that both labels are equal. Based on this match, the entity `wd:Q80871` should replace all the appearances of the placeholder `sbj_1` in the current `SPARQL` query that is being constructed.

```
standard_filling(annotations, slots, query_template)
    sparql_query = query_template
    for s_label, placeholder in slots
        if placeholder has been used or placeholder not in sparql_query
            skip
        if placeholder is a <num> or <str_value>
            replace placeholder in sparql_query for the s_label value
        if placeholder is an <entity_type> (e.g. <sbj_1>)
            for e_label, entity in annotations
                if s_label equal to e_label
                    replace placeholder in sparql_query for the entity value
                    break
        mark placeholder and/or entity as used, if applies
    return sparql_query
```

Listing 3.4: Standard Filling algorithm.

Some problems can arise from this algorithm, which are mainly due to previous systems passing incorrect or incomplete information (e.g. not all the entity annotations are identified, or the Sequence Tagger tag some mentions incorrectly). Consequently, some placeholders may not be filled by the standard filling process. In such cases, a second filling process, called Force Filling, is performed to address these issues.

Force Filling aims to fill all slots in a best-effort manner, and is performed based on some assumptions and heuristics. The first assumption is that since we usually deal with short questions, the number of entities to be identified and placeholders to be filled tend to be low, not having more than three spots to fill in average. We also assume that most of the mistakes made by the Sequence Tagger are due to incorrect tagging. Also, multiple word labels from annotations and from the Sequence Tagger might differ but if they share one or more words, it is more likely that both labels refer to the same value. Lastly, we assume that if one spot is left to be filled, and one entity is left to be used, it is likely that that entity should be placed there even if the slot label and the entity label are not the same. Considering all that, the mistakes made by previous systems can be mitigated by using simple heuristic rules.

To summarize, a similar filling process to the one shown in Listing 3.4 is executed, but adding the following rules:

- If any slot label is contained in an entity label (or vice versa), it counts as a match.
- When comparing entity type placeholders, if the slot placeholder is the same entity type (subject or object) or is assigned to the same query triple (e.g. the first query triple pattern on the `SPARQL` query) as some of the remaining Query Template placeholders,

it counts as a match.

- Finally, if there is any remaining entity that was not identified by the Sequence Tagger, and there are still placeholders to fill in the Query Template, the entities will be inserted in order until all spots have been filled (where the order is defined by the entity identifier number).

As an example, let us assume we have the outputs illustrated in Figure 3.5 for the question *"How many Nobel Prizes has Gabriela Mistral won?"*, which requires a count operation. The expected slots state that the subject of the first query triple should be filled with the entity of the label *"Gabriela Mistral"*, while the object of the second query triple should be filled with the entity of the label *"Nobel Prizes"*. However, it may happen that a label is incorrectly associated with the wrong query triple (case 1), or the wrong entity type (case 2). In the first case, the *"Nobel Prizes"* entity would not be filled in the Standard Slot Filling stage due to the `<obj_2>` not existing in the Query Template, but it would be recognized in the Force Slot Filling stage because it will identify that the remaining slot in the Query Template (`<obj_1>`) is the same entity type as the slot incorrectly labeled (`<obj_2>`). The same happens in the second case, where the incorrect label (`<obj_1>`) is associated with the same query triple (the first triple) as the remaining slot in the Query Template (`<sbj_1>`).

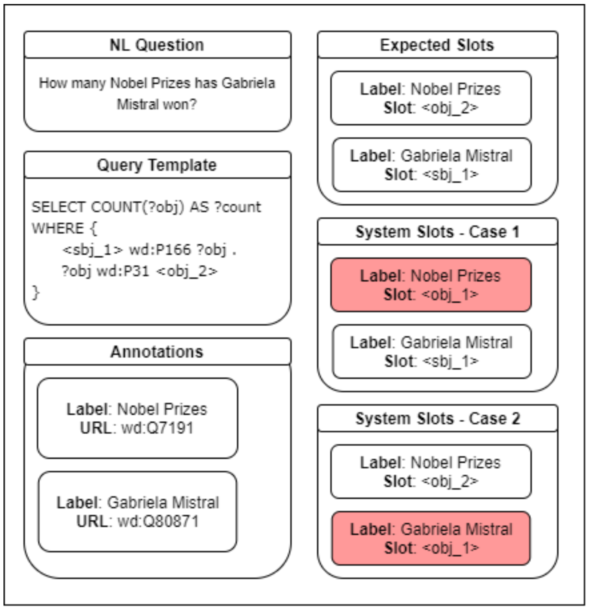


Figure 3.5: Force Filling case examples.

Although more cases can be correctly filled by applying these rules, there are some other cases that are not possible to correctly fill. For example, the Sequence Tagger can identify the expected placeholder but it can wrongly associate each one to an incorrect mention (e.g. wrongly associate the subject entity and object entity of one query triple). A more serious case is when the Sequence tagger provides placeholders that have nothing to do with the ones found in the Query Template, usually because both systems identified a different intention in the given answer. In summary, the success rate of the filling process is strongly affected by the performance of the previous systems that deliver the input for this Slot Filling system.

# Chapter 4

# Experimental Design

We present an overview of the research questions we address in this work along with the details of the experimental design. The implementation of the system, experiments, and datasets used can be found in the following Github repository: `https://github.com/thesemanticwebhero/ElNeuKGQA/`.

## 4.1. Question Answering general overview

The task of Question Answering over Knowledge Graphs is far from being resolved, where one of the main challenges that still remains is answering complex questions. Current approaches rely mostly on hand-crafted rules, achieving decent performance over simple questions (that only require a few query triple patterns), but not for more complex questions. Complex questions are difficult to handle mainly because they may be structured in a more elaborate way, and most of them require the use of more advanced `SPARQL` operations (such as aggregations, ranking, or the use of more complex graph patterns).

Some recent approaches based on Neural Networks have shown some potential for handling more complex questions. These Neural-based approaches usually aim to build the entire `SPARQL` query given a natural question, reducing the problem of Question Answering to Semantic Parsing. The main weakness of these systems is that they are dependent on the vocabulary of the data used for training such models, so even though these systems can generalize to different question forms, they are not able to identify unknown entities that were not included in the training set.

In order to address this problem, we study the effectiveness of an approach that combines Neural Semantic Parsing with Entity Linking through a Slot Filling method proposed in this work. The research questions we propose are the following:

- *Can Entity Linking and Slot Filling improve the performance of Question Answering systems over Knowledge Graphs based on Neural Semantic Parsing?*
- *Which stages of the proposed Question Answering pipeline (Entity Linking, Query Template Generation, Slot Filling) produce the most errors, what types of errors do they produce, and how do they affect the overall performance of the Question Answering system?*

Note that all these questions are addressed specifically in the context of Wikidata as the target Knowledge Graph and for questions in English.

## 4.2. Question Answering Dataset

In this section we describe the dataset used for this work which includes the preprocessing required to improve the quality of the data used for training, validation and testing all of the modules (Query Generation, Entity Linking, and Slot Filling) and the final end-to-end Question Answering System.

First, we describe the revision and cleaning performed over the `LC-QuAD 2` dataset, which is the main source we use to train the Neural Network-based models. Second, we explain the `SPARQL` query encoding used for the training of the Query Template generation and the baseline `SPARQL` Query Generation models. Third, we describe the process for generating the Query Templates used for training the Query Template generation model. Fourth, we describe the same process for the Slot Filling dataset. Finally, we present the normalized dataset format which we designed to handle all the datasets used.

### 4.2.1. LC-QuAD 2 Dataset Cleaning

The `LC-QuAD 2` dataset requires a cleaning process where some cases were discarded or fixed due to various issues in either their question or `SPARQL` query answer. First we mention which cases were discarded according to some problems found in their paraphrased or verbalized question. Next, we describe which cases were discarded according to problems found in their `SPARQL` query. Since there are some queries with an easy fix, we explain briefly those cases.

Given that the questions contained in the `LC-QuAD 2` dataset were built from a two-step question correction (a verbalization and a later paraphrasing), each dataset case contains three representations of the same question: a normalized question, a verbalized question and a paraphrased question. An example is shown in Listing 4.1. Since this process was performed using a crowdsourcing tool, we found issues related with question verbalization (e.g., the human that performed the verbalization didn't understand the normalized question) or loss of the question's semantics through the conversion from normalized to paraphrased questions.

```
normalized question  -> Did {Alexander_Hamilton} {occupation} {lawyer}?
verbalized question  -> Is Alexander Hamilton a lawyer?
paraphrased question -> Did Alexander Hamilton practice law?
```

Listing 4.1: Example of questions contained in one case of the `LC-QuAD 2` dataset.

A case is discarded if both the verbalized and the paraphrased questions present an issue. To define if a question is an invalid verbalization/paraphrase, the following issues are taken into account:

1. The question is null, empty or not applicable (marked as "`NA`").

2. The question has not been corrected, e.g., it still contains brackets from the normalized question, or it is the same question as the normalized version without brackets.

3. The question contains the answer to the previous normalized or verbalized question instead of the verbalization or paraphrasing, respectively.

4. The length of the actual version compared with the previous version is too long or too short, which for us is a way to identify the third issue or an indicator of a question that did not maintain the semantic meaning of its previous version.

5. The question is either too short or too long. We set a minimum of 4 tokens and a maximum of 30, where each token is divided by whitespace characters.

Note that given the large number of questions in the dataset, this validation was not performed manually but rather using a script to identify these issues as best as possible. Thus some undesirable cases may slip into the final preprocessed dataset, though we consider that as an acceptable amount of noise.

Other cases presented issues with their Wikidata `SPARQL` query, where either the query was empty/null, or contained invalid entities or tokens. An entity is considered invalid if it does not present the common Wikidata pattern: a valid prefix (e.g. wd, wdt, etc) along with a valid identifier, which always is composed by a `Q` and a number (e.g. `Q80871`). A few queries contained invalid numbers such as "`t1231`" which are corrected by removing the *t*. Some other queries presented syntax errors or had some clauses missing, which were corrected. Finally, a simple canonicalization process was performed over the variable names in order to follow similar conventions (e.g. use `sbj` and `obj` as base variable names).

Aside from that, some cases were duplicated due to the observation that they contained the same normalized question or the same `SPARQL` query. Those with the same normalized question but different `SPARQL` queries were merged into one case, though just one `SPARQL` query is used later. Other cases with the same `SPARQL` query (regardless of whether their normalized question is the same or not) were discarded.

In summary, from the 30,226 cases contained in the LC-QuAD 2 dataset, 2,520 cases were discarded: of these, 2,478 cases were discarded for having an invalid question or query, and 42 cases were duplicate cases. Note that 52 % of this final preprocessed dataset is composed of complex questions (a deeper explanation of which cases are considered complex can be found in Appendix B). After preprocessing, this dataset is used to train the Neural-based models used in this work.

### 4.2.2. Query Template Dataset

From the cleaned LC-QuAD 2 dataset, a Query Template dataset was created for training the Query Template generator model. More specifically, a Query Template was inferred from each SPARQL query by replacing every subject and object entity in the query with a placeholder. As explained in the System Overview chapter 3.4.2, in order to maintain some degree of the semantic meaning of the entity that is being replaced, we propose using several types of placeholder labels. For example, an entity that was the subject of the first query triple will be replaced by a placeholder labeled as "sbj_1". For numerical or string values, the placeholder would be "num" or "str_value", respectively. This is a different approach to generate Query Templates than the one followed by Yin et al. [197], where instead they replaced entities by one-letter names such as A, B or C.

```
# SPARQL Query
SELECT ?obj WHERE { wd:Q14752155 wdt:P19 ?obj }

# Query Template
SELECT ?obj WHERE { <sbj_1> wdt:P19 ?obj }
```

Listing 4.2: SPARQL query and its Query Template version.

An example can be seen in Listing 4.2, where in the SPARQL query for the question *"Where was Pedro Pascal born?"* the entity Q14752155 from Pedro Pascal was replaced for the sbj_1 placeholder. All queries from the cleaned dataset were able to be transformed into their Query Template version.

Since some cases may have issues with either the verbalized or the paraphrased question, we decided to maintain a one-to-one correspondence between question and query. Although it would be interesting to test whether having many variations of the same question might affect the system performance, we think such an experiment has to be done with a dataset where each case has the same amount of variations (which cannot be guaranteed with this dataset without having to discard an uncertain amount of valid cases). Then, in each case we first use the paraphrased question as the case's question. However, if the paraphrased question is not valid, we use the verbalized question instead (if it is valid).

### 4.2.3. Sequence Labeling Dataset

In order to train the Sequence Labeling model from the Slot Filling module, a dataset was needed. As per the Query Template dataset, the Slot Labeling dataset is based on the cleaned LC-QuAD 2 dataset. The creation of this dataset is divided into two stages.

The first stage consists of identifying the entity-label mapping in terms of which part of the question corresponds to each entity from the ones contained in the SPARQL query. In the example of *"Where was Pedro Pascal born?"*, the label *"Pedro Pascal"* contained in the question should correspond to the entity Q14752155 corresponding to the Chilean actor Pedro Pascal. This mapping identification was performed using the Wikidata labels found in the normalized question provided in the LC-QuAD 2 dataset (see an example in Listing 4.1).

In the second stage, the labels identified previously are associated with the placeholders used for the creation of the Query Template. In this case, the placeholder sbj_1 shown in Listing 4.3 corresponds to the entity Q14752155, therefore the label *"Pedro Pascal"* will be associated with the placeholder sbj_1. We use the BIO label format as the output format for the Sequence Tagger, as shown in Listing 4.3 (the '?' is ignored).

```
# NL Question
Where was Pedro Pascal born ?

# Expected label output
O O B-sbj_1 B-sbj_1 O
```

Listing 4.3: BIO label representation for a Natural Language Question.

Some labels were not able to be associated with their corresponding placeholders due to the question rephrasing performed over the LC-QuAD 2 dataset. The main issue was the discrepancy between the Wikidata label associated with an entity, and its current label in the question (which may be modified due to the rephrasing). As opposed to what we did for the Query Template dataset, we decided to add both verbalized and paraphrased valid cases to compensate for the loss of information. After processing the dataset, we ended up with 36,854 valid cases from the 52,853 initial cases (counting for each case its verbalized and paraphrased version), which is almost 70 % of the dataset.

### 4.2.4. Final Dataset Format

After extracting the Query Templates, the entities, and their entity/placeholder slots, a final LC-QuAD 2 dataset was built containing all this information. In summary, each case possesses a question identifier, the question string, a Wikidata SPARQL query, their entities with the corresponding label, a Query Template deduced from the SPARQL query, and a mapping between the entity labels and the Query Template placeholders.

We proposed a normalized format and delivered a dataset that not only can be used to evaluate systems on the KGQA task, but also over the tasks of Entity Linking, Name Entity Recognition, Slot Filling, Intent Classification, or Semantic Parsing. A sample of the entire dataset can be found in Appendix B.

## 4.3. System Implementation

The system implementation was divided into three main modules which represent each stage described in the *System Overview* chapter 3: Query Generation, Entity Linking, and Slot Filling. These modules were assembled in our proposed Question Answering system, which receives a Natural Language question as an input, and returns a Wikidata `SPARQL` query that can be executed on the Wikidata service endpoint as an output. The Question Answering system only returns a valid `SPARQL` query; otherwise, if one cannot be computed, it will return nothing. The entire system was implemented using Python 3.6.

The implementation of the two variants on the Query Generation module (SPARQL Query Generator and Query Template Generator) was divided into two stages. First the models were trained on Google Colaboratory[1] using the ConvS2s [61] model implementation included in the Fairseq framework[2] and the settings described in the Fairseq Model subsection in the *System Overview* chapter 3.2.3 (we recall that this model had the best performance in the comparison of Yin et al. [197]). After the training phase, a Fairseq Wrapper was implemented in order to import the models and perform the prediction over new data. These wrappers receive the NL question as the input and can return either the best predicted sequence, or a list with the best $N$ predictions (where $N$ can be a number defined by the user).

The various individual Entity Linking systems (DBpedia Spotlight [128], AIDA [198, 89], TAGME [52], OpenTapioca [42]) were implemented using the web services each system has available. Then, an Entity Linking wrapper was created for each system, where they received the NL question as the input returning a set of annotations which contain a Wikidata entity, the label in the question that entity was associated with, and the score the system gives to the respective annotations (which varies across systems). Then, the Precision Priority system and Voting systems were implemented combining the individual Entity Linking wrappers, maintaining the same input–output frame.

Finally, the Slot Filling module implementation followed similar stages to those for the Query Generation module. A Sequence Labeling model is trained using Google Colaboratory and the Flair Sequence Tagger from the Flair framework[3] and the setting described in the *System Overview* chapter 3.4.2. Then, a Flair wrapper was implemented to import the Sequence Labeling model and perform the tagging over new data. This wrapper was used to implement the Slot Filling system, where the input consists of an NL question, a Query

[1] `https://colab.research.google.com/`

[2] `https://fairseq.readthedocs.io/en/latest/`

[3] `https://github.com/flairNLP/flair/`

Template, and the Entity Linking annotations. Then, the output is a SPARQL query built from the given set of values, only if the Slot Filling is successful.

## 4.4. Experiments

Several experiments were conducted for the purpose of getting some insights to address the proposed research questions. Some experiments were performed to determine the optimal setting of the proposed Question Answering (QA) system, in terms of which components are the most appropriate to use (e.g. which Entity Linking configuration delivers the best results), while others are done to validate the performance of each individual module (Query Generation, Entity Linking, Slot Filling) and the end-to-end QA system overall.

### 4.4.1. Datasets

As a summary, the datasets that are used to perform the experiments are the following:

- LC-QuAD 2 [47] (cleaned version), which contains 27,706 questions over Wikidata (from the original 30,000) that are divided into a training set and a validation set in a ratio of 80/20. The training set is only used to train the Query Generator models and the Sequence Tagger model. The validation set is used to evaluate the performance of the end-to-end QA system and its modules. We use a split method that guarantees that all the Wikidata properties are included in at least one case in the training set, thus giving the possibility to a Query Template Generator model to learn from all properties contained in the dataset.

- QALD-7 [178], which is divided into a training set (100 cases) and a test set (50 cases) for Wikidata. Since neither of these datasets are used for training in our experiments, both splits are merged to be used as one dataset to evaluate the performance of the end-to-end QA system and some of its modules. Note that some LC-QuAD 2 templates are based on QALD-7 cases.

- WikiSPARQL, which is a novel dataset generated in this work with the objective of measuring how capable our system is of generalizing to cases that are not necessarily based on the templates used in the LC-QuAD 2 dataset. This dataset consists of 100 cases of manually created questions over Wikidata, and is used to evaluate the performance of the end-to-end QA system and its modules.

An overview of each of the datasets used in this work can be seen in Table 4.1.

At the beginning of this work, we planned to include a mapped version of the DBNQA [81] dataset, with the DBpedia queries being mapped to Wikidata queries. However, we were unable to automatically map more than 12 % of the dataset to Wikidata, losing the variety in query types this dataset provides. Besides, most of the mapped queries did not even

| Dataset | Size | Entities | Properties | Type |
|---|---|---|---|---|
| LC-QuAD 2 | 27,706 | 21,151 | 4,447 | Tran/Valid |
| QALD-7 | 150 | 178 | 100 | Test |
| WikiSPARQL | 100 | 132 | 86 | Test |

Table 4.1: Comparison of Wikidata datasets used for experiments.

return an answer when executed over the Wikidata endpoint. For these reasons, we decided to exclude that dataset from the experiments in this work.

### 4.4.2. Query Template Generation

The Query Template Generator is trained using the LC-QuAD 2 training set, and is evaluated over the training and validation datasets using Perplexity, and BLEU score. Then, the model is evaluated over the LC-QuAD 2 validation, QALD-7 and WikiSPARQL datasets using BLEU score and string-match Accuracy. More details on the hyperparameters and training dataset can be found here in the *System Overview* chapter 3.2.3.

The only point of reference for this model is to use the Baseline Query Generator model, and evaluate which of the two better predicts which Query Template corresponds to each case. The way this comparison can be done is to take the Baseline's SPARQL query output, and convert it into its Query Template version by removing the Wikidata entities that the query contains.

### 4.4.3. Entity Linking

In order to determine which Entity Linking system performs the best, each Individual Entity Linking (IEL) system, along with the Ensemble Entity Linking (EEL) systems proposed in this work, are evaluated over a set of datasets and their results are compared. Note that the results over the IEL systems are also used to determine the priorities that are used in the EEL systems, as explained in the *System Overview* chapter 3.3.3. Aside from that, in the evaluation of the EEL systems proposed (Precision Priority and Voting system), we test the different configurations described in the *System Overview* chapter 3.3.3.

The performance of each Entity Linking system is measured using Precision, Recall and F1-score over their output annotations. Both macro and micro measures are used. First, IEL systems are evaluated over the LC-QuAD 2 validation set in order to select the priority that each IEL system will have for the EEL systems. The priority is determined by comparing the IEL systems' Precision, so an IEL system with higher Precision will have higher priority. Then, EEL systems are also evaluated over the LC-QuAD 2 validation set. After that, both EEL and IEL are evaluated over QALD-7 and our WikiSPARQL datasets.

As a point of reference we calculate the best scenario for an EEL system with an "oracle", where given the results from the IEL systems, the EEL system is capable of choosing only the correct entities. Note that if there is any correct entity that is not recognized for any of the IEL systems, the EEL system itself will not be able to deliver that entity as part of the output annotations. This way, we are setting the performance expectation for an EEL system that should be able to recognize which of the entities delivered are part of the correct set of entities. We will refer to this best EEL system scenario, as the Oracle Entity Linking system (OEL).

### 4.4.4. Sequence Labeling and Slot Filling

The Slot Filling system is evaluated in two phases: evaluation over the Sequence Tagger performance, and evaluation over the Slot Filling algorithm.

First, the Sequence Tagger model is trained using the `LC-QuAD 2` training set, and is evaluated using Precision, Recall, and F1-score over the output labels. Then, the model is evaluated over the `LC-QuAD 2` validation, `QALD-7` and `WikiSPARQL` datasets using the same metrics. Details on the implementation used can be found here in the *System Overview* chapter 3.4.2.

After that, the Slot Filling system is evaluated in terms of how the filling process is performed. Then, using the results obtained from the *Query Template Generation* and the *Entity Linking* experiments, we evaluate the ratio of correct/incorrect cases given the different input cases (e.g. all the incoming input is correct, some of the systems delivered an incorrect output, and so on).

### 4.4.5. Question Answering over Knowledge Graphs

The evaluation of the overall QA system is performed using two approaches: the traditional approach of evaluating answers given by the system [112], and (as used by many Neural-based systems) the approach of evaluating the `SPARQL` query returned [197]. The baseline for both approaches is the `SPARQL` Query generator based on the ConvS2S model that performs the best among the Neural-based models according to the Ying et al.'s work [197].

The first approach consists of evaluating the performance of the QA system using Precision, Recall and F1-score over the system's answers. This approach values more the final output of the system, regardless of how it was retrieved. The second approach consists of evaluating the performance of the QA system using BLEU score and exact string-match Accuracy. This last approach values more the query string used to generate the answer, and how close it is for the expected query. Both approaches are evaluated over the `LC-QuAD 2` validation set, `QALD-7`, and `WikiSPARQL`.

Aside from a general performance evaluation, we study the performance of the system considering not only the best `SPARQL` query answer, but the five `SPARQL` queries with the highest scores given by the system. This can be performed given that the Query Template Generator is capable of returning a list of templates. We believe that this is an interesting analysis given that a possible heuristic to improve the system performance is to look for the correct answer by trying more than one possible SPARQL query, which would work particularly well under the assumption that the input question indeed has some answer, and that many of the incorrect SPARQL query outputs will not generate answers. This analysis is evaluated over the LC-QuAD 2 validation set, QALD-7, and WikiSPARQL.

Then, in order to understand which components of a question and its SPARQL query influence the QA system performance more, an evaluation per template is performed. The same evaluation over the entire dataset is split into partitions corresponding to each LC-QuAD 2 base template. Given that only LC-QuAD 2 is a template-based dataset; this analysis is only performed over that dataset.

Finally, a more granular analysis is conducted in order to understand how much each stage influences the overall results. We compare the correct and incorrect cases of each module (Query Template generation, Entity Linking, and Slot Filling) with the correct/incorrect cases of the QA system, and calculate the ratio of error of each module and how much each one of them contributes to the overall error of the QA system. This analysis is evaluated over the LC-QuAD 2 validation set, QALD-7, and WikiSPARQL.

# Chapter 5

# Results

In this chapter we present the results obtained from the experiments proposed in the *Experimental Design* chapter 4. For each section, we display a table with the results along with a brief discussion for each case. All results are based on models trained on the training set of `LC-QuAD 2`.

## 5.1. Query Template generation

In Table 5.1 we can see the Perplexity and BLEU score values after training the Convolutional Sequence to Sequence model (ConvS2S) for 40 epochs. Note that each epoch outputs a model checkpoint, and these results correspond to the best checkpoint in terms of the model with the lowest loss value over the validation set.

| **System** | **LC-QuAD 2 (train)** | | **LC-QuAD 2 (valid)** | |
|---|---|---|---|---|
| | **Size: 21,394** | | **Size: 5,772** | |
| | **Perplexity** | **BLEU score** | **Perplexity** | **BLEU score** |
| ConvS2S | 1.17 | 91.88 | 1.47 | 73.42 |

Table 5.1: Perplexity and BLEU score after training the ConvS2S model.

Table 5.2 shows the results of the Query Template task evaluated over the `LC-QuAD 2` validation dataset and the other two test datasets. We include the results of the Query Template generator which uses the trained ConvS2S model over `LC-QuAD 2` data. Aside from the evaluation over the best predicted Query Template, we also include an evaluation over the best prediction among the top 5 predictions. We consider the best prediction as the one with the highest BLEU score if compared with the expected answer. The baseline is the ConvS2S model trained over complete `SPARQL` queries (as done by Yin et al. [197]), but adding an extra layer that removes the entities from the output `SPARQL` queries, in a similar way as was done for generating the actual LC-QuAD 2 Query Template dataset. The training results of the baseline are shown later in this chapter.

| **System** | **LC-QuAD 2 (valid)** | | **QALD-7** | | **WikiSPARQL** | |
|---|---|---|---|---|---|---|
| | **Size: 21,394** | | **Size: 5,772** | | **Size: 100]** | |
| | **BLEU score** | **Accuracy (%)** | **BLEU score** | **Accuracy (%)** | **BLEU score** | **Accuracy (%)** |
| QTG - Top 1 | 65.18 | 34.27 | 20.29 | 0 | 20.12 | 0 |
| QTG - Top 5 | 76.86 | 49.53 | 22.58 | 0.67 | 23.10 | 0 |
| Baseline | 63.06 | 29.82 | 19.72 | 0.67 | 20.31 | 0 |

Table 5.2: BLEU score and Accuracy for the Query Template generation task.

According to the results of Tables 5.1 and 5.2, we see that the Query Template generation has a better performance than the Baseline in both BLEU score and Accuracy. If we look at the best 5 for the LC-QuAD 2 validation dataset, we find there are better results. On the other hand, we see that the models trained on the LC-QuAD 2 training set do not generalize well to other datasets.

| **Performance analysis per template (QT generator vs Baseline) - LC-QuAD 2 valid** | | | | | | |
|---|---|---|---|---|---|---|
| **ID** | **Template type** | **Size** | **% Dataset** | **QTG Accuracy** | **Baseline Accuracy** | **Diff Accuracy** |
| 1 | ask_one_fact | 112 | 1.94 % | 51.79 % | 17.86 % | 33.93 % |
| 2 | ask_one_fact_with_filter | 362 | 6.27 % | 45.03 % | 42.82 % | 2.21 % |
| 3 | ask_two_facts | 113 | 1.96 % | 30.97 % | 7.96 % | 23.01 % |
| 4 | count_one_fact_object | 126 | 2.18 % | 25.40 % | 23.02 % | 2.38 % |
| 5 | count_one_fact_subject | 182 | 3.15 % | 9.89 % | 3.85 % | 6.04 % |
| 6 | rank_instance_of_type_one_fact | 78 | 1.35 % | 32.05 % | 34.62 % | **-2.57 %** |
| 7 | rank_max_instance_of_type_two_facts | 68 | 1.18 % | 7.35 % | 7.35 % | 0.00 % |
| 8 | rank_min_instance_of_type_two_facts | 70 | 1.21 % | 14.29 % | 8.57 % | 5.72 % |
| 9 | select_object_instance_of_type | 392 | 6.79 % | 23.72 % | 23.47 % | 0.25 % |
| 10 | select_object_using_one_statement_property | 583 | 10.10 % | 50.94 % | 40.82 % | 10.12 % |
| 11 | select_one_fact_object | 78 | 1.35 % | 7.69 % | 10.26 % | **-2.57 %** |
| 12 | select_one_fact_subject | 288 | 4.99 % | 5.90 % | 9.72 % | **-3.82 %** |
| 13 | select_one_qualifier_value_and_object_using_one_statement_property | 136 | 2.36 % | 52.94 % | 48.53 % | 4.41 % |
| 14 | select_one_qualifier_value_using_one_statement_property | 631 | 10.93 % | 52.14 % | 39.78 % | 12.36 % |
| 15 | select_subject_instance_of_type | 424 | 7.35 % | 25.00 % | 20.75 % | 4.25 % |
| 16 | select_subject_instance_of_type_contains_word | 286 | 4.95 % | 70.98 % | 61.19 % | 9.79 % |
| 17 | select_subject_instance_of_type_starts_with | 285 | 4.94 % | 77.54 % | 74.04 % | 3.50 % |
| 18 | select_two_answers | 191 | 3.31 % | 16.23 % | 9.95 % | 6.28 % |
| 19 | select_two_facts_left_subject | 393 | 6.81 % | 13.23 % | 12.98 % | 0.25 % |
| 20 | select_two_facts_right_subject | 412 | 7.14 % | 14.08 % | 17.72 % | **-3.64 %** |
| 21 | select_two_facts_subject_object | 389 | 6.74 % | 31.88 % | 31.11 % | 0.77 % |
| 22 | select_two_qualifier_values_using_one_statement_property | 173 | 3.00 % | 26.59 % | 24.28 % | 2.31 % |
| Total | | 5772 | 100.00 % | 34.67 % | 29.82 % | 4.85 % |

Table 5.3: Performance comparison per template between Query Template generator and Baseline.

We present in Table 5.3 a more granular comparison between our proposed Query Template generator and the baseline by dividing the results into the 22 base templates used for generating the LC-QuAD 2 dataset (more details on each base template can be found in Appendix B). We note that, even though our Query Template generator performs better in many cases, there are some others where the Baseline performs better. A possible explanation for this difference is that the Baseline still maintains an advantage in terms of the amount of information it keeps in the entities used in training versus the placeholders used to replace those entities for generating the training data for our Query Template generator. While the Baseline can identify more effective entity categories (e.g. people, locations, organizations), the placeholder only can maintain information about entity location in the query (to which query pattern it belongs) or entity type (subject or object).

In summary, we see that the same ConvS2S model can perform as well or better for the task of generating Query Templates as it does for SPARQL query generation, obtaining scores of BLEU and Accuracy equal or greater than the ones obtained from the baseline. On the other hand, we still identified a certain degree of overfitting in the model (similar to the results obtained by Yin et al. [197]), thus not being able to perform well on the proposed test datasets (QALD-7 and WikiSPARQL). However, we think the metrics used for measuring performance here (BLEU score and Accuracy) do not quantify the equivalences between Query Templates properly. For example, queries in the test data could have a different ordering in their query triples than the generator uses to build Query Templates.

## 5.2. Entity Linking

First, we calculate the performance of each Individual Entity Linking (IEL) system over the validation LC-QuAD 2 dataset. These results were used to determine the priority each IEL system will have when being used by the Ensemble Entity Linking (EEL) systems. As mentioned before, the priority of each IEL system is determined by their Precision results where more precise systems have higher priority. Therefore, according to the results in Table 5.4, the priority used is the following: AIDA, OpenTapioca, TAGME, and DBpedia Spotlight.

| **LCQUAD2 (valid): 5772 questions** | | | | | | |
|---|---|---|---|---|---|---|
| **System** | **Micro** | | | **Macro** | | |
| | **Recall** | **Precision** | **F1-score** | **Recall** | **Precision** | **F1-score** |
| AIDA | 31.3 | 72.5 | 43.7 | 30.5 | 38.5 | 33.1 |
| OpenTapioca | 32.0 | 61.8 | 42.2 | 30.2 | 34.9 | 31.4 |
| TAGME | 59.5 | 25.0 | 35.2 | 59.4 | 29.5 | 37.4 |
| DBpedia Spotlight | 52.7 | 20.8 | 29.9 | 52.5 | 23.3 | 30.8 |

Table 5.4: Performance of the Individual Entity Linking systems over LC-QuAD 2 validation.

Next, we tested the two EEL systems proposed in this work: the Precision Priority (PPrior) system and the Voting system. For each EEL system we tried different heuristics proposed to improve performance: vary the number of entities in the output, filter stopwords, or apply tiebreak using IEL system scores. In the case of varying the number of entities, we tried including all entities, reducing output size to either a fixed value[1] or a variable amount defined by the number of placeholders contained in the Query Template for each case.

The baseline (Oracle system) combines all the results of the IEL systems and evaluates what is the best performance an EEL system could achieve if it could "guess" the correct entities among the IEL system answers in each case; in other words, the baseline indicates the best possible result given the results of the IEL systems.

The variants for both EEL systems are shown in Table 5.5, where we tried different combinations of the proposed heuristics. Here we evaluate the results over the LC-QuAD 2 validation dataset, resulting in the Voting system having a slight better performance for most variations.

| **LCQUAD2 (valid): 5772 questions** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **System** | **Output size** | **Stopwords filter** | **Tiebreak** | **Micro** | | | **Macro** | | |
| | | | | **Recall** | **Precision** | **F1-score** | **Recall** | **Precision** | **F1-score** |
| Precision prior | All | True | – | 68.6 | 19.5 | 30.3 | 68.4 | 23.9 | 33.6 |
| Precision prior | 3 | True | True | 63.8 | 32.1 | 42.7 | 64.1 | 32.7 | 41.9 |
| Precision prior | + | False | False | 58.2 | 33.8 | 42.8 | 57.6 | 44.4 | 48.2 |
| Precision prior | + | True | False | 59.3 | 34.8 | 43.9 | 58.8 | 45.6 | 49.3 |
| Precision prior | + | True | True | 55.1 | 55.3 | 55.2 | 54.4 | 54.4 | 54.4 |
| Voting system | All | True | – | 68.6 | 19.5 | 30.3 | 68.4 | 23.9 | 33.6 |
| Voting system | 3 | True | True | 63.7 | 32.1 | 42.7 | 64.0 | 32.7 | 41.9 |
| Voting system | + | False | False | 57.6 | 43.7 | 49.7 | 57.0 | 50.4 | 52.5 |
| Voting system | + | True | False | 58.3 | 44.5 | 50.5 | 57.7 | 51.1 | 53.2 |
| Voting system | + | True | True | 56.2 | 56.4 | 56.3 | 55.4 | 55.4 | 55.4 |
| Baseline: Oracle system | | | | 68.6 | 70.4 | 69.5 | 68.4 | 68.4 | 68.4 |

Table 5.5: Results of the Ensemble Entity Linking systems over LC-QuAD 2 validation.

Note that when including all entities the results of the PPrior system and Voting system are identical, though the ranking scores assigned to each entity might differ. Furthermore, the variants that include all entities in the output results reach a better Recall, which is consistent with these variants including more entities that are potentially part of the expected answers.

The effect of decreasing the amount of entities returned for each case is a trade-off between Recall and Precision: while the Recall tends to decrease, we note a major increase in Precision. While the case with a fixed number of three maintains most of the correct answers without degrading Precision much, the case where the number varies (denoted as "+" in Table 5.5) presents the worst performance but the best Precision. The filter of stopwords helped a small amount in both Recall and Precision, whereas the use of tiebreak greatly benefits Precision while slightly degrading Recall.

Finally, we evaluate both IEL systems and EEL systems over the test datasets: QALD-7, and our proposed WikiSPARQL dataset. For each EEL system we include the three variants

[1] Since most SPARQL queries included in our datasets have at most three entities, we set that value as the fixed size.

| QALD 7: 150 questions | | | | | | |
|---|---|---|---|---|---|---|
| System | Micro | | | Macro | | |
| | Recall | Precision | F1-score | Recall | Precision | F1-score |
| AIDA | 41.2 | 79.1 | 54.2 | 43.7 | 54.0 | 47.1 |
| OpenTapioca | 33.2 | 60.3 | 42.8 | 34.2 | 38.9 | 34.2 |
| TAGME | 58.3 | 31.5 | 40.9 | 62.0 | 36.0 | 43.1 |
| DBpedia Spotlight | 62.6 | 31.4 | 41.8 | 65.7 | 32.3 | 41.1 |
| Precision Prior (All) | 74.4 | 27.7 | 40.4 | 76.6 | 32.5 | 43.6 |
| Precision Prior (3) | 69.7 | 35.5 | 47.0 | 72.2 | 37.3 | 47.5 |
| Precision Prior (+) | 58.8 | 59.0 | 58.9 | 60.6 | 60.6 | 60.6 |
| Voting system (All) | 74.4 | 27.7 | 40.4 | 76.6 | 32.5 | 43.6 |
| Voting system (3) | 68.7 | 35.0 | 46.4 | 71.3 | 36.9 | 47.0 |
| Voting system (+) | 61.1 | 61.4 | 61.3 | 63.0 | 63.0 | 63.0 |
| Oracle system | 74.4 | 76.2 | 75.3 | 76.6 | 76.8 | 76.6 |

Table 5.6: Results of the Entity Linking systems over QALD-7 (train+test).

| WikiSPARQL: 100 questions | | | | | | |
|---|---|---|---|---|---|---|
| System | Micro | | | Macro | | |
| | Recall | Precision | F1-score | Recall | Precision | F1-score |
| AIDA | 23.3 | 75.4 | 35.5 | 27.3 | 34.2 | 29.5 |
| OpenTapioca | 14.0 | 51.2 | 22.0 | 18.8 | 20.6 | 18.9 |
| TAGME | 60.7 | 37.0 | 46.0 | 59.3 | 40.3 | 45.5 |
| DBpedia Spotlight | 60.0 | 34.9 | 44.1 | 57.5 | 38.9 | 43.1 |
| Precision Prior (All) | 75.3 | 32.7 | 45.6 | 72.5 | 38.4 | 47.6 |
| Precision Prior (3) | 69.3 | 38.0 | 49.1 | 67.3 | 41.4 | 49.0 |
| Precision Prior (+) | 52.7 | 49.4 | 51.0 | 51.2 | 52.2 | 51.6 |
| Voting system (All) | 75.3 | 32.7 | 45.6 | 72.5 | 38.4 | 47.6 |
| Voting system (3) | 70.7 | 38.7 | 50.0 | 68.1 | 42.1 | 49.7 |
| Voting system (+) | 56.7 | 53.1 | 54.8 | 53.2 | 54.2 | 53.5 |
| Oracle system | 75.3 | 76.4 | 75.8 | 76.5 | 76.7 | 76.6 |

Table 5.7: Results of the Entity Linking systems over our proposed WikiSPARQL dataset.

in output size used for Table 5.5. Tables 5.6 and 5.7 display the result for the QALD-7, and WikiSPARQL, respectively.

From the results of Tables 5.6 and 5.7 we can see that overall the Voting system is the system with the best F1 score. The EEL system variants with the highest Recall and Precision maintain that tendency over all datasets. In this case, it seems that the systems with variable output size fall behind compared to the oracle, where the best performing system is around 0.15 to 0.2 points behind the oracle results in terms of F1-score.

## 5.3. Sequence Labeling and Slot Filling

The performance results of the Flair Sequence Tagger are shown in Table 5.8. In this case, the Precision and Recall are measured with respect to the pairs `<slot, label>`, where the slot is a placeholder of Query Template and the label is a substring of a natural language question.

| Dataset | Number of cases | Micro | | | Macro | | |
|---|---|---|---|---|---|---|---|
| | | Recall | Precision | F1-score | Recall | Precision | F1-score |
| LC-QuAD 2 (valid) | 5772 | 66.9 | 62.0 | 64.4 | 66.9 | 63.8 | 64.8 |
| QALD-7 | 150 | 43.6 | 37.0 | 40.0 | 45.3 | 45.4 | 44.8 |
| WikiSPARQL | 100 | 29.8 | 22.7 | 25.8 | 31.8 | 30.9 | 31.0 |

Table 5.8: Results of the Flair Sequence Tagger.

Next, we measure how the proposed filling method performs. In this case, the Precision and Recall are measured with respect to the pairs `<slot, entity>`, where the slot remains the same and the entity corresponds to the ones contained in a `SPARQL` query. We test the standard filling method, and the force filling method we proposed to address the Sequence Tagger errors. These results are shown in Table 5.9 and 5.10, respectively. Note that we are using the Entity Linking results corresponding to the Voting system returning all entities in the output. We chose to return all entities to see how each filling method performs when there are incorrect entities. Also, we use the Voting system given that it presents better performance than the PPrior system according to the results shown in the *Entity Linking* section 5.2, as well as for the results shown below in the *SPARQL Query generation* section 5.4.

| Dataset | Number of cases | Micro | | | Macro | | |
|---|---|---|---|---|---|---|---|
| | | Recall | Precision | F1-score | Recall | Precision | F1-score |
| LC-QuAD 2 (valid) | 5772 | 45.7 | 61.4 | 52.4 | 46.7 | 54.2 | 49.2 |
| QALD-7 | 150 | 27.9 | 52.6 | 36.4 | 35.2 | 39.0 | 36.3 |
| WikiSPARQL | 100 | 13.2 | 26.3 | 17.6 | 19.8 | 22.3 | 20.5 |

Table 5.9: Results of the Slot Filling system using the standard filling method.

| Dataset | Number of cases | Micro | | | Macro | | |
|---|---|---|---|---|---|---|---|
| | | Recall | Precision | F1-score | Recall | Precision | F1-score |
| LC-QuAD 2 (valid) | 5772 | 49.2 | 51.0 | 50.1 | 50.3 | 52.7 | 50.5 |
| QALD-7 | 150 | 39.2 | 31.3 | 32.1 | 40.8 | 34.0 | 35.9 |
| WikiSPARQL | 100 | 17.0 | 16.5 | 16.7 | 20.6 | 16.8 | 17.7 |

Table 5.10: Results of the Slot Filling system using the force filling method.

The number of correct cases for both slot filling methods is lower than the correctly labeled cases of the Sequence Tagger. This is expected since many incorrect cases might occur due to the errors given by the Entity Linking system. On the other hand, the standard filling method shows more Precision while the force filling method brings more Recall. This is also expected since the force filling method adds more labels that might bring more correct answers, along with more incorrect ones. As opposed to the case for Entity Linking systems, we prefer a filling method with higher Recall in order to generate more valid SPARQL queries.

## 5.4. SPARQL Query Generation

The training results for the Baseline of the SPARQL Query generation task are shown in Table 5.11. We note similar results as the ones obtained in Yin et al [197], where there is notable overfitting for the training data.

| **System** | **LC-QuAD 2 (train)** | | **LC-QuAD 2 (valid)** | |
|---|---|---|---|---|
| | **Size: 21934** | | **Size: 5772** | |
| | **Perplexity** | **BLEU score** | **Perplexity** | **BLEU score** |
| Baseline | 1.19 | 98.35 | 3.20 | 60.39 |

Table 5.11: Perplexity and BLEU score after training the SPARQL Query Generator baseline.

Next, the results of the Entity Linking Neural Question Answering (ElNeuQA) system proposed in this work are displayed in Table 5.12. In this case, we try four different variants where we chose the best variant of both PPrior and Voting systems and both slot filling methods. These variations are tested over the LC-QuAD 2 validation dataset. We also output results picking the best among the top 5 predicted results per case over both variants that utilized the Force filling method. In this context, the criterion to select the best results offers the highest BLEU score.

| **System** | **Entity Linking system** | **Filling method** | **LC-QuAD 2 (valid)** | |
|---|---|---|---|---|
| | | | **Size: 5772** | |
| | | | **BLEU score** | **Accuracy (%)** |
| ElNeuQA | PPrior (All) | Standard | 58.01 | 11.83 |
| ElNeuQA | PPrior (+) | Standard | 57.36 | 10.41 |
| ElNeuQA | Voting (All) | Standard | 58.47 | 12.72 |
| ElNeuQA | Voting (+) | Standard | 57.60 | 10.88 |
| ElNeuQA | PPrior (All) | Force | 59.16 | 13.70 |
| ElNeuQA | PPrior (+) | Force | 57.95 | 10.90 |
| **ElNeuQA** | **Voting (All)** | **Force** | **59.34** | **13.96** |
| ElNeuQA | Voting (+) | Force | 58.13 | 11.31 |
| ElNeuQA - Top 5 | PPrior (All) | Force | 68.16 | 20.32 |
| ElNeuQA - Top 5 | Voting (All) | Force | 68.43 | 20.70 |
| Baseline | | | 51.50 | 3.27 |

Table 5.12: Comparison of performance for the SPARQL Query generator when varying the EEL system and filling method.

The results obtained for `LC-QuAD 2` do not transfer to the other test datasets. As seen in Table 5.13, our system is not capable of generalizing the results obtained from training over other types of datasets given that the types of queries contained in test datasets are not based on template generation, thus making it difficult to generate an exact match. We may also be generating equivalent `SPARQL` queries (i.e. that return the same answers if executed), and the way we are evaluating these results does not consider `SPARQL` query equivalences.

| **System** | **LC-QuAD 2 (valid)** | | **QALD-7** | | **WikiSPARQL** | |
|---|---|---|---|---|---|---|
| | **Size: 5772** | | **Size: 150** | | **Size: 100** | |
| | **BLEU score** | **Accuracy (%)** | **BLEU score** | **Accuracy (%)** | **BLEU score** | **Accuracy (%)** |
| ElNeuQA - Top 1 | 59.34 | 13.96 | 20.14 | 0 | 20.48 | 0 |
| ElNeuQA - Top 5 | 68.43 | 20.70 | 21.99 | 0 | 22.15 | 0 |
| Baseline | 51.50 | 3.27 | 19.09 | 0 | 18.80 | 0 |

Table 5.13: BLEU score and Accuracy for the `SPARQL` Query generation task.

| **Performance analysis per template (ElNeuQA vs Baseline) - LC-QuAD 2 valid** | | | | | | |
|---|---|---|---|---|---|---|
| **ID** | **Template type** | **Size** | **% Dataset** | **ElNeuQA Accuracy** | **Baseline Accuracy** | **Diff Accuracy** |
| 1 | ask_one_fact | 112 | 1.94 % | 36.61 % | 0.00 % | 36.61 % |
| 2 | ask_one_fact_with_filter | 362 | 6.27 % | 20.44 % | 0.83 % | 19.61 % |
| 3 | ask_two_facts | 113 | 1.96 % | 15.93 % | 0.00 % | 15.93 % |
| 4 | count_one_fact_object | 126 | 2.18 % | 20.63 % | 8.73 % | 11.90 % |
| 5 | count_one_fact_subject | 182 | 3.15 % | 9.34 % | 0.55 % | 8.79 % |
| 6 | rank_instance_of_type_one_fact | 78 | 1.35 % | 11.54 % | 17.95 % | **-6.41 %** |
| 7 | rank_max_instance_of_type_two_facts | 68 | 1.18 % | 0.00 % | 1.47 % | **-1.47 %** |
| 8 | rank_min_instance_of_type_two_facts | 70 | 1.21 % | 8.57 % | 0.00 % | 8.57 % |
| 9 | select_object_instance_of_type | 392 | 6.79 % | 4.34 % | 6.12 % | **-1.78 %** |
| 10 | select_object_using_one_statement_property | 583 | 10.10 % | 19.55 % | 0.17 % | 19.38 % |
| 11 | select_one_fact_object | 78 | 1.35 % | 7.69 % | 1.28 % | 6.41 % |
| 12 | select_one_fact_subject | 288 | 4.99 % | 3.82 % | 4.17 % | **-0.35 %** |
| 13 | select_one_qualifier_value_and_object_using_one_statement_property | 136 | 2.36 % | 47.06 % | 17.65 % | 29.41 % |
| 14 | select_one_qualifier_value_using_one_statement_property | 631 | 10.93 % | 20.29 % | 0.16 % | 20.13 % |
| 15 | select_subject_instance_of_type | 424 | 7.35 % | 5.42 % | 7.31 % | **-1.89 %** |
| 16 | select_subject_instance_of_type_contains_word | 286 | 4.95 % | 13.64 % | 2.10 % | 11.54 % |
| 17 | select_subject_instance_of_type_starts_with | 285 | 4.94 % | 14.04 % | 16.84 % | **-2.80 %** |
| 18 | select_two_answers | 191 | 3.31 % | 14.14 % | 3.66 % | 10.48 % |
| 19 | select_two_facts_left_subject | 393 | 6.81 % | 8.91 % | 0.00 % | 8.91 % |
| 20 | select_two_facts_right_subject | 412 | 7.14 % | 9.95 % | 0.00 % | 9.95 % |
| 21 | select_two_facts_subject_object | 389 | 6.74 % | 14.14 % | 0.00 % | 14.14 % |
| 22 | select_two_qualifier_values_using_one_statement_property | 173 | 3.00 % | 8.67 % | 2.31 % | 6.36 % |
| **Total** | | 5772 | 100.00 % | 13.96 % | 3.27 % | 10.69 % |

Table 5.14: Performance comparison per template for the `SPARQL` Query generation task.

The same analysis per template performed over the Query Template generation task are done for the SPARQL query generation task. In Table 5.14 the results in terms of Accuracy are displayed for each LC-QuAD 2 base template. We can see that there are more cases than before where our proposed system performed better than the Baseline, especially due to the inability of the latter system to generate a single valid query for some templates. Among those cases, the ask_one_fact template is the most noticeable case since it should be the simplest one. The main reason for these cases to occur is that the Baseline system fails to identify the relevant entities due to not having seen them during training time. As we will discuss later, the vocabulary dependence of the Baseline plays a major role in the worse performance of this system.

The proposed system manages to achieve better Accuracy in most cases. The cases that present the best results are the ASK type queries with one query triple (ask_one_fact) and the SELECT queries that utilize one property statement and one qualifier property (select_one_ qualifier_value_and_object_using_one_statement_property), both of which require only one entity. One noticeable case where our system does not present a good performance is the rank type query that requires two query triples, where no query was successfully generated. As opposed to the best performing cases, these rank queries were among the most complex cases since they require 2 entities and 3 properties to be correctly predicted; thus a single error would make the entire query incorrect.

| **Template** | **Entities** | **Slots** | **ElNeuQA** | | **Baseline** | |
|---|---|---|---|---|---|---|
| | | | **Number of cases** | **Dataset %** | **Number of cases** | **Dataset %** |
| O | O | O | 806 | 13.96 % | 189 | 3.27 % |
| O | O | X | 229 | 3.97 % | 74 | 1.28 % |
| O | X | O | 0 | 0.00 % | 0 | 0.00 % |
| O | X | X | 967 | 16.74 % | 1458 | 25.26 % |
| X | O | O | 903 | 15.64 % | 158 | 2.74 % |
| X | O | X | 1415 | 24.51 % | 149 | 2.58 % |
| X | X | O | 0 | 0.00 % | 0 | 0.00 % |
| X | X | X | 1452 | 25.17 % | 3744 | 64.86 % |
| **Total** | | | 5772 | 100.00 % | 5772 | 100.00 % |

Table 5.15: SPARQL query generation comparison results with baseline divided by error cases.

Finally, in order to identify the most common sources of errors, we divided the results depending on the correct/incorrect answers expected for each module (Query Template generator, Entity Linking, Slot Filling). A comparison between our proposal (using the best performing settings) and the baseline is presented in Table 5.15. For each case, a circle means the system performed correctly and a cross is the opposite. For the Query Template generation task, a correct case is measured using the string-match Accuracy. For Entity Linking we use Recall, meaning that we count a case as correct if it identifies at least all expected entities, regardless of its Precision (understanding that this is the minimum requirement to build a valid SPARQL query). For Slot Filling we consider it correct if all slot–entity pairs are correctly identified. Note that a correct SPARQL can only be generated if these three items deliver a correct answer.

The most noticeable comparison between both systems comes from the major difference in terms of vocabulary dependency for entities. While only 47.6 % of the incorrect cases were caused by entity recognition error for our system, in the Baseline almost 94 % of the incorrect cases were caused due to not identifying all relevant entities. The error delivered from the Query Template generation was similar for both our system and the baseline, with 65.32 % and 70.2 % incorrect cases respectively.

The performance of the Slot Filling system is strongly impacted by the results delivered by the Entity Linking system. This can be inferred from the fact that there is no case where the slots were correctly identified while the entities were not. Aside from that, we notice that the Slot Filling system correctly filled 78 % of the 1035 cases where both templates and entities are correctly identified (i.e. cases from the first and second row of Table 5.15).

| **Analysis per template for each module of the ElNeuQA system** | | | | | | |
|---|---|---|---|---|---|---|
| **ID** | **Template** | **Dataset** | **Accuracy** | **Template** | **Entity** | **Slot** |
| 1 | ask_one_fact | 1.90 % | 36.61 % | 51.79 % | 84.48 % | 83.67 % |
| 2 | ask_one_fact_with_filter | 6.30 % | 20.44 % | 45.03 % | 59.38 % | 77.89 % |
| 3 | ask_two_facts | 2.00 % | 15.93 % | 30.97 % | 62.86 % | 81.82 % |
| 4 | count_one_fact_object | 2.20 % | 20.63 % | 25.40 % | 84.38 % | 96.30 % |
| 5 | count_one_fact_subject | 3.20 % | 9.34 % | 9.89 % | 94.44 % | 100.00 % |
| 6 | rank_instance_of_type_one_fact | 1.40 % | 11.54 % | 32.05 % | 40.00 % | 90.00 % |
| 7 | rank_max_instance_of_type_two_facts | 1.20 % | 0.00 % | 7.35 % | 20.00 % | 0.00 % |
| 8 | rank_min_instance_of_type_two_facts | 1.20 % | 8.57 % | 14.29 % | 80.00 % | 75.00 % |
| 9 | select_object_instance_of_type | 6.80 % | 4.34 % | 23.72 % | 22.58 % | 80.95 % |
| 10 | select_object_using_one_statement_property | 10.10 % | 19.55 % | 50.94 % | 68.35 % | 70.37 % |
| 11 | select_one_fact_object | 1.40 % | 7.69 % | 7.69 % | 100.00 % | 100.00 % |
| 12 | select_one_fact_subject | 5.00 % | 3.82 % | 5.90 % | 76.47 % | 84.62 % |
| 13 | select_one_qualifier_value_and_object_using_one_statement_property | 2.40 % | 47.06 % | 52.94 % | 90.28 % | 98.46 % |
| 14 | select_one_qualifier_value_using_one_statement_property | 10.90 % | 20.29 % | 52.14 % | 77.04 % | 61.54 % |
| 15 | select_subject_instance_of_type | 7.30 % | 5.42 % | 25.00 % | 26.42 % | 82.14 % |
| 16 | select_subject_instance_of_type_contains_word | 5.00 % | 13.64 % | 70.98 % | 28.49 % | 76.47 % |
| 17 | select_subject_instance_of_type_starts_with | 4.90 % | 14.04 % | 77.54 % | 29.21 % | 76.92 % |
| 18 | select_two_answers | 3.30 % | 14.14 % | 16.23 % | 100.00 % | 87.10 % |
| 19 | select_two_facts_left_subject | 6.80 % | 8.91 % | 13.23 % | 69.23 % | 97.22 % |
| 20 | select_two_facts_right_subject | 7.10 % | 9.95 % | 14.08 % | 74.14 % | 95.35 % |
| 21 | select_two_facts_subject_object | 6.70 % | 14.14 % | 31.88 % | 52.42 % | 84.62 % |
| 22 | select_two_qualifier_values_using_one_statement_property | 3.00 % | 8.67 % | 26.59 % | 54.35 % | 60.00 % |
| **Total** | | 100.00 % | 13.96 % | 34.67 % | 57.12 % | 78.00 % |

Table 5.16: ElNeuQA modules correct cases per template.

A more granular analysis per template is provided in Table 5.16. The purpose is to understand how the Entity Linking and Slot Filling perform per template when the Query Template is correctly generated. We provide the SPARQL Query accuracy from Table 5.14

and the percentage of correct Query Templates (Template), similar to what is shown in Table 5.3. In the case of the entities (Entity), we only include the cases with a correct Query Template (though the overall results per template do not vary much). For the slots (Slot), we only count the cases where the Query Template and entities are correctly identified [2].

When the template requires one entity (e.g. templates 4, 5, 10, 11, 12, 13, 18) the Entity Linking system obtains the best results. There are some exceptions (e.g. templates 9, 15, 16, 17) that expect more generic entities like human (`Q5`), city (`Q515`), writer (`Q36180`), etc; which are usually aligned with triples using the `wdt:P31` property (instance of). The problem with those types of entities might be that they don't score high for Entity Linking systems in terms of relevance, given that some terms can be perceived as more common, thus not being considered meaningful. The same happens with the rank-type templates (i.e. templates 7 and 8) that require at least one "instance of" entity, which would further explain the poor performance for those cases. Lastly, in other cases that require more than one entity (e.g. templates 1, 14), Entity Linking succeeds in recognizing all entities in a good amount of cases (where the Query Template is correctly identified).

In regards to the Slot Filling system performance, results from Table 5.16 reassure us that when the input is properly provided, this module can deliver good results overall. When there is only one entity to replace, the filling process is executed almost perfectly (e.g. templates 4, 5, 11, 19, 20). We believe the incorrect cases might come from choosing the wrong entity [3], which also applies for templates that require more than one entity. Slot filling has lower success for templates with two to three triples (e.g. templates 7, 8, 22) that contain placeholders such as `obj_2` or `obj_3`, which can be explained by the low prevalence these placeholders had in the training dataset. An interesting case are the templates that contain non-entity placeholders, i.e. numeric or string values, where around 3 of 4 cases are filled correctly (see templates 2, 16, 17).

## 5.5. Question Answering over Knowledge Graphs

We evaluate the results of the `SPARQL` queries output by our QA system proposal (using the best settings in terms of `SPARQL` Query generation performance) over Wikidata, and compare those results with the proposed Baseline. We show in Table 5.17 the results for both systems on the KGQA task. We chose to display only the macro measures given the difference of answer sizes for each case, where we wanted to weigh each case equally. Here we noticed that even though neither our proposal nor the baseline were capable of generating the expected `SPARQL` queries for the test datasets, some generated `SPARQL` queries were capable of obtaining answers among the expected results.

[2] We make this differentiation because we are only interested in illustrating the correctness of the Slot Filling system when the input (i.e. Query Template and entities) is well provided. We don't expect this module to perform correctly otherwise.

[3] Let us remember that even though the Entity Linking system might identify the correct entity, it may also include some garbage entities that add noise to the filling process

We also evaluate the performance looking at the best 5 predictions per case, where a general improvement is noticed along all datasets. In this case, the best prediction is the first query that returns a non-empty answer when being executed in the Wikidata endpoint. Note that this approach is not effective with `ASK` type questions, since there is always a non-empty answer (True or False) or in cases where the correct query may not return results.

| **System** | **LC-QuAD 2 (valid)** | | | **QALD-7** | | | **WikiSPARQL** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Size: 5772** | | | **Size: 150** | | | **Size: 100** | | |
| | **Recall** | **Precision** | **F1** | **Recall** | **Precision** | **F1** | **Recall** | **Precision** | **F1** |
| ElNeuQA - Top 1 | 27.0 | 26.9 | 26.9 | 9.3 | 9.3 | 9.3 | 12.9 | 12.9 | 12.9 |
| ElNeuQA - Top 5 | 30.0 | 29.6 | 29.6 | 12.7 | 12.7 | 12.7 | 11.9 | 11.9 | 11.9 |
| Baseline | 16.6 | 16.4 | 16.4 | 7.3 | 7.3 | 7.3 | 6.0 | 5.5 | 5.3 |

Table 5.17: Macro measure results for the Question Answering task.

Finally, an analysis over the different `LC-QuAD 2` templates is presented in Table 5.18. We noticed that our proposal does better in almost every case compared with the baseline. The cases with the best performance include templates like `ASK` queries, `SELECT` queries that include property statements and qualifiers.

Another aspect to notice is that there is no clear correlation between the performance and the amount of cases per template, meaning that a greater amount of cases for one template to use in the training stage does not necessarily come with an increase in performance. A clear example are the `COUNT` queries: while the `count_one_fact_object` template has fewer examples than the `count_one_fact_subject`, our system performs worse with the latter template.

Lastly, the complexity of each template does not necessarily predict the performance of the Question Answering system in terms of Macro F1. Queries derived from complex templates such as the `rank_max_instance_of_type_two_facts` template present better performance than simpler templates such as the `select_subject_instance_of_type` template. There might be other variables that affect performance per template that we are not considering here. For example, there might be entities or properties used for each template that can be more complex to identify or are not very common, thus making it more difficult to predict those cases.

| **Performance analysis per template (ElNeuQA vs Baseline) - LC-QuAD 2 valid** | | | | | | |
|---|---|---|---|---|---|---|
| **ID** | **Template type** | **Size** | **%** | **ElNeuQA** | **Baseline** | **Diff** |
| | | | **Dataset** | **Macro F1** | **Macro F1** | **Macro F1** |
| 1 | ask_one_fact | 112 | 1.94 % | 54.46 | 16.07 | 38.39 |
| 2 | ask_one_fact_with_filter | 362 | 6.27 % | 45.86 | 34.25 | 11.60 |
| 3 | ask_two_facts | 113 | 1.96 % | 60.18 | 30.97 | 29.20 |
| 4 | count_one_fact_object | 126 | 2.18 % | 21.43 | 11.11 | 10.32 |
| 5 | count_one_fact_subject | 182 | 3.15 % | 10.44 | 2.20 | 8.24 |
| 6 | rank_instance_of_type_one_fact | 78 | 1.35 % | 18.72 | 31.79 | **-13.08** |
| 7 | rank_max_instance_of_type_two_facts | 68 | 1.18 % | 30.98 | 25.29 | 5.69 |
| 8 | rank_min_instance_of_type_two_facts | 70 | 1.21 % | 31.43 | 23.43 | 8.00 |
| 9 | select_object_instance_of_type | 392 | 6.79 % | 16.07 | 20.38 | **-4.32** |
| 10 | select_object_using_one_statement_property | 583 | 10.10 % | 33.28 | 14.82 | 18.46 |
| 11 | select_one_fact_object | 78 | 1.35 % | 14.10 | 6.41 | 7.69 |
| 12 | select_one_fact_subject | 288 | 4.99 % | 11.79 | 12.15 | **-0.36** |
| 13 | select_one_qualifier_value_and_object_using_one_statement_property | 136 | 2.36 % | 48.61 | 20.11 | 28.50 |
| 14 | select_one_qualifier_value_using_one_statement_property | 631 | 10.93 % | 31.41 | 11.89 | 19.52 |
| 15 | select_subject_instance_of_type | 424 | 7.35 % | 17.30 | 19.87 | **-2.57** |
| 16 | select_subject_instance_of_type_contains_word | 286 | 4.95 % | 27.19 | 16.41 | 10.79 |
| 17 | select_subject_instance_of_type_starts_with | 285 | 4.94 % | 29.99 | 26.34 | 3.65 |
| 18 | select_two_answers | 191 | 3.31 % | 22.43 | 7.33 | 15.10 |
| 19 | select_two_facts_left_subject | 393 | 6.81 % | 17.02 | 9.03 | 7.99 |
| 20 | select_two_facts_right_subject | 412 | 7.14 % | 20.54 | 10.84 | 9.70 |
| 21 | select_two_facts_subject_object | 389 | 6.74 % | 28.02 | 14.91 | 13.11 |
| 22 | select_two_qualifier_values_using_one_statement_property | 173 | 3.00 % | 26.59 | 16.18 | 10.40 |
| **Total** | | 5772 | 100.00 % | 26.90 | 16.40 | 10.50 |

Table 5.18: Performance comparison per template for the Question Answering task.

# Chapter 6

# Conclusions

In this work, we address the Question Answering over Knowledge Graphs task, which, given a natural language question, consists of generating the expected answer when evaluated on the KG. In particular, we proposed a Question Answering system that generates a `SPARQL` query expressing the question, and that combines Entity Linking systems with a Neural Machine Translation model to address the dependency on the vocabulary used for training the NMT model when generating `SPARQL` queries.

The `SPARQL` query generation process was divided into several stages. First, we trained a Convolutional Sequence to Sequence model and we used it to generate a Query Template, which is a `SPARQL` query with its entities replaced by placeholders. Second, relevant entities were extracted using an Ensemble Entity Linking system, which combines many Individual Entity Linking systems to increase the amount of entities identified. Two variants of Ensemble Entity Linking system were proposed: a Precision Priority system and a Voting system. Finally, we proposed a Slot Filling system to fill entities into the Query Template, which relies on a Sequence Tagger (based on the state-of-art in Named Entity Recognition) and a proposed filling algorithm, giving the final `SPARQL` query as a result.

We conducted various experiments to compare the performance of our system with a proposed baseline, which includes an overall comparison for the tasks of `SPARQL` query generation and Question Answering, and also an analysis for each one of the three sub-tasks involved in our proposal (Query Template Generation, Entity Linking, and Slot Filling).

According to the **hypothesis** proposed in this work, we showed that by combining Entity Linking and Neural Semantic Parsing through a Slot Filling system, it is possible to obtain better performance in the Question Answering over Knowledge Graph task. This is according to the proposed metrics used to compare the system: BLEU score and Accuracy for `SPARQL` query generation, and Precision, Recall and F1 score for Question Answering. We found that this increase in performance comes mainly from the reduction of errors in the system due to not identifying the correct entities required to build the SPARQL query.

It is important to remember that our hypothesis was explored in the context of Wikidata for questions in the English language. Nevertheless, the approach we follow should be generalizable to other Knowledge Graphs and languages. The conclusions based on the hypothesis also allow us to determine that our **specific objective** of building a Question Answering system over Wikidata for English questions was accomplished, though this system is obviously subject to further improvement in future.

At the time that this work was developed, and to the best of our knowledge, the models described by Yin et al. [197] were the state-of-the-art for KGQA systems based on Neural Machine Translation models. They validate each model over DBpedia datasets. In our case, we took the model they stated has better performance to compare with our system. Moreover, we validate our results using LC-QuAD 2 (a more complex dataset in terms of questions variations and SPARQL query complexity); QALD-7 (which is also used to evaluate Information Retrieval-based QA systems), and WikiSPARQL (a new dataset with cases that require new types of operations). That being said, we think our **general objective** of improving the state-of-the-art was accomplished.

The last thing to mention are the **limitations** we identified in our work. The first issue is related with the performance shown in the Query Template generation task. According to our results, most of the cases where the SPARQL query was built incorrectly were due to the Query Template being incorrect. Even though in this work we could not identify the main cause of the poor performance for this task, we think it might be caused by different reasons: either the LC-QuAD 2 dataset contained cases that are inherently difficult to respond even for an SPARQL expert, or it does not contain enough cases to allow the NMT model to generalize across templates. As a consequence, the SPARQL queries our proposed KGQA system can generate are strongly limited by the type of question found in the LC-QuAD 2 training set. For example, our system may be able to generate questions that require counting, but given that the only COUNT type query included in the LC-QuAD 2 dataset uses only one fact, questions that require counting and use more than one fact will not be correctly interpreted.

The second limitation involves our proposed Slot Filling system using a Sequence Tagger to label the placeholders in the Query Template, which is an "ad-hoc" solution for this work. This means that if we want to train our proposal over another dataset, we will have to adapt that dataset to be utilized to train the Sequence Tagger. That is one of the reasons we also proposed a normalized dataset format, so the Sequence Tagger dataset construction could be simplified.

## 6.1. Relevance and Contributions

First, we propose a benchmark for comparing Question Answering systems that work over Wikidata. By cleaning the LC-QuAD 2 to have less noisy cases and a normalized format, we reduce the amount of work others will have to spend in having a ready to use dataset to either train Neural-based models or evaluating their KGQA systems. Moreover, we deliver a new dataset that includes new `SPARQL` operations and patterns. These datasets can not only be used to evaluate KGQA systems, but also to compare Entity Linking systems and Slot Filling systems.

Second, we contribute with a new Question Answering system over Wikidata for questions in English. The implementation comes from our proposed three-stage pipeline that addresses the `SPARQL` Query generation process. This pipeline can be used to implement Question Answering systems focused on other Knowledge Graphs or languages.

As a side effect, we propose an approach to boost the performance of Entity Linking systems when dealing with questions that have little to no context: Ensemble Entity Linking systems. We proposed two variants on how to combine results from individual Entity Linking systems to deliver a more complete response. The Precision priority system shows how to improve performance by prioritizing results from more precise systems.

The type of questions our system can address are directly tied to the types of questions contained in the `LC-QuAD 2` dataset. Some examples are boolean questions (including cases that use literal numeric values), and questions that require use of string operations, counting, property statements, or ranking. Even though not all query variants are considered, this is a big step forward in terms of the type of queries that are supported, since none of the previous Neural-based models support cases we did consider by using the `LC-QuAD 2` dataset (e.g. use of numeric values or literal strings).

In terms of overall relevance, we deliver new insights on how to address the Question Answering over Knowledge Graphs task. In particular, we proposed an approach that aims to address the vocabulary dependency that Neural-based models might suffer from, and show how there is a direct impact on Neural-based KGQA systems' performance.

## 6.2. Future work

The Question Answering over Knowledge Graphs task is still a challenging problem despite there being plenty of work in this field. There are individual components of our system that can be improved but also new things to try or add to our system.

One thing is to work on improving the training dataset by increasing its size and adding the `SPARQL` query variants the `LC-QuAD 2` dataset lacks. We think a bigger dataset would help the Query Template generator model have a better learning process, and the addition of new variants would let the model address a wider range of questions. Furthermore, we should focus on questions that are asked more frequently by the users of Wikidata. Then, a possible next step would be to implement a crowdsourcing tool that lets users ask questions for our system, thus identifying common patterns or categories that can be materialized in new query base templates. Lastly, it would be interesting to add new operations in the training examples (e.g. property paths, conditional operations, optional patterns or unions of results).

Another aspect that could be improved is the learning architecture we are using for the Query Template generator, where we could also try either of the other models used in Yin et al.'s work, or try new architectures from the state-of-the-art for Semantic Parsing or Machine Translation. One other component which might help better estimate the model performance in practice, would be to follow a *k*-fold cross-validation during the training phase [9]. Aside from that, we could also propose new logical forms for representing Query Templates. For example, we can condense some common query patterns (e.g. the syntax pattern to see whether a string is contained in an entity label is always the same), or use program-based representations.

Even though our proposed Slot Filling system manages to perform well enough, we would like to have a system that reduces the amount of incorrect slots as much as possible. One proposal would be to test other placeholder classifications that capture different dimensions of entities, e.g. tag entities as humans, organizations, places, etc. Another approach that we may follow is to include the placeholder labeling during the Query Template generation stage, perhaps by adapting the Pointer Network architecture [181], which aims to address the issue of vocabulary dependency by adding pointers in the output sequence that map to labels contained in the text being processed.

To work more on the Entity Linking stage, we see a potential in the Voting Entity Linking system we proposed in this work. We think that the performance of this system can be improved horizontally, i.e., we can add further Individual Entity Linking systems to improve the results of the voting scheme. Moreover, we can add Entity Linking systems focused on specific categories (e.g. sports, medicine, music, etc), so more rare entities can be identified more easily.

As we mentioned before, our approach should be generalizable to other languages or Knowledge Graph domains. Another possible step is to evaluate our system over other languages such as Spanish, French, or German, or to try to generate `SPARQL` queries for other Knowledge Graphs such as DBpedia. To give an idea, the main changes that would be required are essentially to provide a dataset that either changes the input language of the questions or the input Knowledge Graph and `SPARQL` queries, used for both training and evaluation.

# Bibliografía

[1] Asma Ben Abacha and Pierre Zweigenbaum. Medical question answering: translating medical questions into sparql queries. In Gang Luo, Jiming Liu, and Christopher C. Yang, editors, *ACM International Health Informatics Symposium, IHI '12, Miami, FL, USA, January 28-30, 2012*, pages 41–50. ACM, 2012.

[2] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and Processing. W3C Recommendation, October 2008. `https://www.w3.org/TR/rdfa-syntax/`.

[3] Peter Adolphs, Martin Theobald, Ulrich Schäfer, Hans Uszkoreit, and Gerhard Weikum. YAGO-QA: Answering Questions by Structured Knowledge Queries. In *Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC 2011), Palo Alto, CA, USA, September 18-21, 2011*, pages 158–161. IEEE Computer Society, 2011.

[4] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. In Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pages 54–59. Association for Computational Linguistics, 2019.

[5] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual String Embeddings for Sequence Labeling. In Emily M. Bender, Leon Derczynski, and Pierre Isabelle, editors, *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 1638–1649. Association for Computational Linguistics, 2018.

[6] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Commun. ACM*, 26(11):832–843, 1983.

[7] Guozhong An. The effects of adding noise during backpropagation training on a gene-

ralization performance. *Neural computation*, 8(3):643–674, 1996.

[8] Jacob Andreas, Andreas Vlachos, and Stephen Clark. Semantic Parsing as Machine Translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pages 47–52. The Association for Computer Linguistics, 2013.

[9] Davide Anguita, Luca Ghelardoni, Alessandro Ghio, Luca Oneto, and Sandro Ridella. The'K'in K-fold Cross Validation. In *ESANN*, pages 441–446, 2012.

[10] Yoav Artzi, Nicholas FitzGerald, and Luke S. Zettlemoyer. Semantic Parsing with Combinatory Categorial Grammars. In *51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, Proceedings of the Conference Tutorial Abstracts, 4-9 August 2013, Sofia, Bulgaria*, page 2. The Association for Computer Linguistics, 2013.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[12] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for Sembanking. In Stefanie Dipper, Maria Liakata, and Antonio Pareja-Lora, editors, *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, LAW-ID@ACL 2013, August 8-9, 2013, Sofia, Bulgaria*, pages 178–186. The Association for Computer Linguistics, 2013.

[13] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. Constraint-Based Question Answering with Knowledge Graph. In Nicoletta Calzolari, Yuji Matsumoto, and Rashmi Prasad, editors, *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pages 2503–2514. ACL, 2016.

[14] Dave Beckett and Brian McBride. RDF/XML Syntax Specification (Revised). W3C Recommendation, February 2004. `https://www.w3.org/TR/rdf-syntax-grammar/`.

[15] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. RDF 1.1 Turtle – Terse RDF Triple Language. W3C Recommendation, February 2014. `https://www.w3.org/TR/turtle/`.

[16] Jonathan Berant and Percy Liang. Semantic Parsing via Paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1415–

1425. The Association for Computer Linguistics, 2014.

[17] Tim Berners-Lee. Linked Data. W3C Design Issues, July 2006. From `https://www.w3.org/DesignIssues/LinkedData.html`; retr. 2010/10/27.

[18] Tim Berners-Lee and Dan Connolly. Notation3 (N3): A readable RDF syntax. W3C Team Submission, March 2011. `https://www.w3.org/TeamSubmission/n3/`.

[19] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[20] Mark Birbeck and Shane McCarron. CURIE Syntax 1.0 – A syntax for expressing Compact URIs. W3C Recommendation, January 2009. `https://www.w3.org/TR/curie/`.

[21] Christopher M Bishop et al. *Neural networks for pattern recognition.* Oxford university press, 1995.

[22] Roi Blanco, Giuseppe Ottaviano, and Edgar Meij. Fast and Space-Efficient Entity Linking for Queries. In Xueqi Cheng, Hang Li, Evgeniy Gabrilovich, and Jie Tang, editors, *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM 2015, Shanghai, China, February 2-6, 2015*, pages 179–188. ACM, 2015.

[23] Christoph Böhm, Markus Freitag, Arvid Heise, Claudia Lehmann, Andrina Mascher, Felix Naumann, Vuk Ercegovac, Mauricio A. Hernández, Peter Haase, and Michael Schmidt. GovWILD: integrating open government data for transparency. In Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 321–324. ACM, 2012.

[24] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10–12, 2008*, pages 1247–1250, 2008.

[25] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Measuring semantic similarity between words using web search engines. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 757–766. ACM, 2007.

[26] Abdelghani Bouziane, Djelloul Bouchiha, Noureddine Doumi, and Mimoun Malki. Question Answering Systems: Survey and Trends. *Procedia Computer Science*, 73:366

– 375, 2015. International Conference on Advanced Wireless Information and Communication Technologies (AWICT 2015).

[27] Dan Brickley, R.V. Guha, and Andrew Layman. Resource Description Framework (RDF) Schemas. W3C Working Draft, April 1998. `https://www.w3.org/TR/1998/WD-rdf-schema-19980409/`.

[28] Razvan C. Bunescu and Marius Pasca. Using Encyclopedic Knowledge for Named entity Disambiguation. In Diana McCarthy and Shuly Wintner, editors, *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy*. The Association for Computer Linguistics, 2006.

[29] Sebastian Burgstaller-Muehlbacher, Andra Waagmeester, Elvira Mitraka, Julia Turner, Tim Putman, Justin Leong, Chinmay Naik, Paul Pavlidis, Lynn Schriml, Benjamin M Good, and Andrew I Su. Wikidata as a semantic framework for the Gene Wiki initiative. *Database*, 2016, 03 2016.

[30] Ruichu Cai, Boyan Xu, Zhenjie Zhang, Xiaoyan Yang, Zijian Li, and Zhihao Liang. An Encoder-Decoder Framework Translating Natural Language to Database Queries. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 3977–3983. ijcai.org, 2018.

[31] Nilesh Chakraborty, Denis Lukovnikov, Gaurav Maheshwari, Priyansh Trivedi, Jens Lehmann, and Asja Fischer. Introduction to Neural Network based Approaches for Question Answering over Knowledge Graphs. *CoRR*, abs/1907.09361, 2019.

[32] Jinmiao Chen and Narendra S. Chaudhari. Capturing Long-Term Dependencies for Protein Secondary Structure Prediction. In Fuliang Yin, Jun Wang, and Chengan Guo, editors, *Advances in Neural Networks - ISNN 2004, International Symposium on Neural Networks, Dalian, China, August 19-21, 2004, Proceedings, Part II*, volume 3174 of *Lecture Notes in Computer Science*, pages 494–500. Springer, 2004.

[33] Jason P. C. Chiu and Eric Nichols. Named Entity Recognition with Bidirectional LSTM-CNNs. *CoRR*, abs/1511.08308, 2015.

[34] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi, editors, *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111. Association for Computational Linguistics, 2014.

[35] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014.

[36] Marco Cornolti, Paolo Ferragina, and Massimiliano Ciaramita. A framework for benchmarking entity-annotation systems. In Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo Baeza-Yates, and Sue B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 249–260. International World Wide Web Conferences Steering Committee / ACM, 2013.

[37] Marco Cornolti, Paolo Ferragina, Massimiliano Ciaramita, Stefan Rüd, and Hinrich Schütze. A Piggyback System for Joint Entity Mention Detection and Linking in Web Queries. In Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 567–578. ACM, 2016.

[38] Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In Jason Eisner, editor, *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 708–716. ACL, 2007.

[39] Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In Jason Eisner, editor, *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 708–716. ACL, 2007.

[40] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Identification of the Question Focus: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*. European Language Resources Association, 2010.

[41] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language Modeling with Gated Convolutional Networks. *CoRR*, abs/1612.08083, 2016.

[42] Antonin Delpeuch. OpenTapioca: Lightweight Entity Linking for Wikidata. *CoRR*,

abs/1904.09131, 2019.

[43] Dennis Diefenbach, Vanessa López, Kamal Deep Singh, and Pierre Maret. Core techniques of question answering systems over knowledge bases: a survey. *Knowl. Inf. Syst.*, 55(3):529–569, 2018.

[44] Li Dong and Mirella Lapata. Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[45] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From Data Fusion to Knowledge Fusion. *CoRR*, abs/1503.00302, 2015.

[46] Mark Dredze, Paul McNamee, Delip Rao, Adam Gerber, and Tim Finin. Entity Disambiguation for Knowledge Base Population. In Chu-Ren Huang and Dan Jurafsky, editors, *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*, pages 277–285. Tsinghua University Press, 2010.

[47] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, volume 11779 of *Lecture Notes in Computer Science*, pages 69–78. Springer, 2019.

[48] Leo Egghe. Untangling Herdan's law and Heaps' law: Mathematical and informetric arguments. *J. Assoc. Inf. Sci. Technol.*, 58(5):702–709, 2007.

[49] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandecic. Introducing Wikidata to the Linked Data Web. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2014.

[50] Anthony Fader, Luke S. Zettlemoyer, and Oren Etzioni. Paraphrase-Driven Learning for Open Question Answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 1608–1618. The Association for Computer Linguistics,

2013.

[51] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. SPARQL 1.1 Protocol. W3C Recommendation, March 2013. `https://www.w3.org/TR/sparql11-protocol/`.

[52] Paolo Ferragina and Ugo Scaiella. TAGME: on-the-fly annotation of short text fragments (by Wikipedia entities). In Jimmy Huang, Nick Koudas, Gareth J. F. Jones, Xindong Wu, Kevyn Collins-Thompson, and Aijun An, editors, *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, pages 1625–1628. ACM, 2010.

[53] Óscar Ferrández, Christian Spurk, Milen Kouylekov, Iustin Dornescu, Sergio Ferrández, Matteo Negri, Rubén Izquierdo, David Tomás, Constantin Orasan, Guenter Neumann, Bernardo Magnini, and José Luis Vicedo González. The QALL-ME Framework: A specifiable-domain multilingual Question Answering architecture. *J. Web Semant.*, 9(2):137–145, 2011.

[54] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2962–2970, 2015.

[55] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir R. Radev. Improving Text-to-SQL Evaluation Methodology. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 351–360. Association for Computational Linguistics, 2018.

[56] Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In Kevin Knight, Hwee Tou Ng, and Kemal Oflazer, editors, *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA*, pages 363–370. The Association for Computer Linguistics, 2005.

[57] André Freitas, Joao Gabriel Oliveira, Edward Curry, Seán O'Riain, and Joao Carlos Pereira da Silva. Treo: combining entity-search, spreading activation and semantic relatedness for querying linked data. In *Proc. of 1st Workshop on Question Answering over Linked Data (QALD-1) at the 8th Extended Semantic Web Conference (ESWC 2011)*, 2011.

[58] André Freitas, João Gabriel Oliveira, Seán O'Riain, João Carlos Pereira da Silva, and Edward Curry. Querying linked data graphs using semantic relatedness: A vocabulary independent approach. *Data Knowl. Eng.*, 88:126–141, 2013.

[59] Bin Fu, Yunqi Qiu, Chengguang Tang, Yang Li, Haiyang Yu, and Jian Sun. A Survey on Complex Question Answering over Knowledge Base: Recent Advances and Challenges. *CoRR*, abs/2007.13069, 2020.

[60] Octavian-Eugen Ganea, Marina Ganea, Aurelien Lucchi, Carsten Eickhoff, and Thomas Hofmann. Probabilistic bag-of-hyperlinks model for entity linking. In *Proceedings of the 25th International Conference on World Wide Web*, pages 927–938, 2016.

[61] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR, 2017.

[62] Daniel Gerber and Axel-Cyrille Ngonga Ngomo. Extracting Multilingual Natural-Language Patterns for RDF Predicates. In Annette ten Teije, Johanna Völker, Siegfried Handschuh, Heiner Stuckenschmidt, Mathieu d'Aquin, Andriy Nikolov, Nathalie Aussenac-Gilles, and Nathalie Hernandez, editors, *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, volume 7603 of *Lecture Notes in Computer Science*, pages 87–96. Springer, 2012.

[63] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Comput.*, 12(10):2451–2471, 2000.

[64] Cristina Giannone, Valentina Bellomaria, and Roberto Basili. A HMM-based Approach to Question Answering against Linked Data. In Pamela Forner, Roberto Navigli, Dan Tufis, and Nicola Ferro, editors, *Working Notes for CLEF 2013 Conference , Valencia, Spain, September 23-26, 2013*, volume 1179 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[65] Daniel Gildea and Daniel Jurafsky. Automatic Labeling of Semantic Roles. *Comput. Linguistics*, 28(3):245–288, 2002.

[66] Alfio Massimiliano Gliozzo and Aditya Kalyanpur. Predicting Lexical Answer Types in Open Domain QA. *Int. J. Semantic Web Inf. Syst.*, 8(3):74–88, 2012.

[67] Alfio Massimiliano Gliozzo and Aditya Kalyanpur. Predicting Lexical Answer Types in Open Domain QA. *Int. J. Semantic Web Inf. Syst.*, 8(3):74–88, 2012.

[68] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010.

[69] Christine Golbreich and Evan K. Wallace. OWL 2 Web Ontology Language: New Features and Rationale. W3C Recommendation, December 2012. `https://www.w3.org/TR/owl2-new-features/`.

[70] Jing Gong, Chong Feng, Yong Liu, Ge Shi, and Heyan Huang. Collective Entity Linking on Relational Graph Model with Mentions. In Maosong Sun, Xiaojie Wang, Baobao Chang, and Deyi Xiong, editors, *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data - 16th China National Conference, CCL 2017, - and - 5th International Symposium, NLP-NABD 2017, Nanjing, China, October 13-15, 2017, Proceedings*, volume 10565 of *Lecture Notes in Computer Science*, pages 159–171. Springer, 2017.

[71] Jan Grant and David Beckett. RDF Test Cases. W3C Recommendation, February 2004. `https://www.w3.org/TR/rdf-testcases/`.

[72] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012.

[73] Alex Graves. Generating Sequences With Recurrent Neural Networks. *CoRR*, abs/1308.0850, 2013.

[74] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[75] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE, 2013.

[76] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052. IEEE, 2005.

[77] Sherzod Hakimov, Christina Unger, Sebastian Walter, and Philipp Cimiano. Applying Semantic Parsing to Question Answering Over Linked Data: Addressing the Lexical Gap. In Chris Biemann, Siegfried Handschuh, André Freitas, Farid Meziane, and

Elisabeth Métais, editors, *Natural Language Processing and Information Systems - 20th International Conference on Applications of Natural Language to Information Systems, NLDB 2015 Passau, Germany, June 17-19, 2015 Proceedings*, volume 9103 of *Lecture Notes in Computer Science*, pages 103–109. Springer, 2015.

[78] Scott A. Hale. Multilinguals and Wikipedia Editing. *CoRR*, abs/1312.0976, 2013.

[79] Xianpei Han, Le Sun, and Jun Zhao. Collective entity linking in web text: a graph-based method. In Wei-Ying Ma, Jian-Yun Nie, Ricardo Baeza-Yates, Tat-Seng Chua, and W. Bruce Croft, editors, *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 765–774. ACM, 2011.

[80] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 Query Language. W3C Recommendation, March 2013. `https://www.w3.org/TR/sparql11-query/`.

[81] Ann-Kathrin Hartmann, Edgard Marx, and Tommaso Soru. Generating a Large Dataset for Neural Question Answering over the DBpedia Knowledge Base. 2018.

[82] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015.

[83] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[84] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space (1st Edition)*, volume 1 of *Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool, 2011. Available from `http://linkeddatabook.com/editions/1.0/`.

[85] Ivan Herman, Ben Adida, Mark Birbeck, and Shane McCarron. RDFa 1.1 Primer – Second Edition. W3C Working Group Note, August 2013. `https://www.w3.org/TR/rdfa-primer/`.

[86] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[87] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[88] Johannes Hoffart, Stephan Seufert, Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. KORE: keyphrase overlap relatedness for entity disambiguation. In Xue-wen Chen, Guy Lebanon, Haixun Wang, and Mohammed J. Zaki, editors, *21st ACM In-*

*ternational Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 545–554. ACM, 2012.

[89] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 782–792. ACL, 2011.

[90] Konrad Höffner, Sebastian Walter, Edgard Marx, Ricardo Usbeck, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. Survey on challenges of Question Answering in the Semantic Web. *Semantic Web*, 8(6):895–920, 2017.

[91] Aidan Hogan. Linked Data & the Semantic Web Standards. In Andreas Harth, Katja Hose, and Ralf Schenkel, editors, *Linked Data Management*, pages 3–48. Chapman and Hall/CRC, 2014.

[92] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing Deep Neural Networks with Logic Rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[93] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR*, abs/1508.01991, 2015.

[94] Heng Ji, Ralph Grishman, Hoa Trang Dang, Kira Griffitt, and Joe Ellis. Overview of the TAC 2010 knowledge base population track. In *Third text analysis conference (TAC 2010)*, volume 3, pages 3–3, 2010.

[95] Robin Jia and Percy Liang. Data Recombination for Neural Semantic Parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[96] Kam-Chuen Jim, C Lee Giles, and Bill G Horne. An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on neural networks*, 7(6):1424–1438, 1996.

[97] Tim Johnson. Natural Language Computing: The Commercial Applications. *Knowl. Eng. Rev.*, 1(3):11–23, 1984.

[98] Karen Spärck Jones. A statistical interpretation of term specificity and its application

in retrieval. *J. Documentation*, 60(5):493–502, 2004.

[99] Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. Automating Formalization by Statistical and Semantic Parsing of Mathematics. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2017.

[100] Aishwarya Kamath and Rajarshi Das. A Survey on Semantic Parsing. In *1st Conference on Automated Knowledge Base Construction, AKBC 2019, Amherst, MA, USA, May 20-22, 2019*, 2019.

[101] Michael Kifer. Rule Interchange Format: The Framework. In Diego Calvanese and Georg Lausen, editors, *Web Reasoning and Rule Systems, Second International Conference, RR 2008, Karlsruhe, Germany, October 31-November 1, 2008. Proceedings*, volume 5341 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2008.

[102] Petri Koistinen and Lasse Holmström. Kernel regression and backpropagation training with noise. *Advances in Neural Information Processing Systems*, 4:1033–1039, 1991.

[103] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[104] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural Architectures for Named Entity Recognition. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 260–270. The Association for Computational Linguistics, 2016.

[105] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, February 1999. `https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

[106] Yann LeCun, Leon Bottou, G Orr, and Klaus-Robert Muller. Efficient backprop. *Neural Networks: Tricks of the Trade. New York: Springer*, 1998.

[107] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia – A large-scale, multilingual knowledge base extracted

from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[108] Xin Li and Dan Roth. Learning Question Classifiers. In *19th International Conference on Computational Linguistics, COLING 2002, Howard International House and Academia Sinica, Taipei, Taiwan, August 24 - September 1, 2002*, 2002.

[109] Chen Liang, Jonathan Berant, Quoc V. Le, Kenneth D. Forbus, and Ni Lao. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. *CoRR*, abs/1611.00020, 2016.

[110] Chen Liang, Jonathan Berant, Quoc V. Le, Kenneth D. Forbus, and Ni Lao. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 23–33. Association for Computational Linguistics, 2017.

[111] Vanessa López, Miriam Fernández, Enrico Motta, and Nico Stieler. PowerAqua: Supporting users in querying and exploring the Semantic Web. *Semantic Web*, 3(3):249–265, 2012.

[112] Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. Evaluating Question Answering over Linked Data. *Web Semantics Science Services And Agents On The World Wide Web*, 21:3–13, 2013.

[113] Vanessa López, Victoria S. Uren, Marta Sabou, and Enrico Motta. Is Question Answering fit for the Semantic Web?: A survey. *Semantic Web*, 2(2):125–155, 2011.

[114] Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. Joint Entity Recognition and Disambiguation. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 879–888. The Association for Computational Linguistics, 2015.

[115] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421. The Association for Computational Linguistics, 2015.

[116] Fabiano Ferreira Luz and Marcelo Finger. Semantic Parsing Natural Language into SPARQL: Improving Target Language Representation with Neural Attention. *CoRR*, abs/1803.04329, 2018.

[117] Xuezhe Ma and Eduard H. Hovy. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[118] Xuezhe Ma and Fei Xia. Unsupervised Dependency Parsing with Transferring Distribution via Parallel Guidance and Entropy Regularization. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1337–1348. The Association for Computer Linguistics, 2014.

[119] Behrooz Mahasseni, Michael Lam, and Sinisa Todorovic. Unsupervised Video Summarization with Adversarial LSTM Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2982–2991. IEEE Computer Society, 2017.

[120] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *The Semantic Web – ISWC 2018 – 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II*, pages 376–394, 2018.

[121] Frank Manola, Eric Miller, and Brian McBride. RDF Primer. W3C Recommendation, February 2004. `https://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

[122] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Comput. Linguistics*, 19(2):313–330, 1993.

[123] Anca Marginean. Question answering over biomedical linked data with Grammatical Framework. *Semantic Web*, 8(4):565–580, 2017.

[124] Jose L. Martinez-Rodriguez, Aidan Hogan, and Ivan Lopez-Arevalo. Information Extraction meets the Semantic Web: A Survey. *Semantic Web*, 11(2):255–335, 2020.

[125] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, February 2004. `https://www.w3.org/TR/owl-features/`.

[126] Paul McNamee, James Mayfield, Dawn J. Lawrie, Douglas W. Oard, and David S. Doermann. Cross-Language Entity Linking. In *Fifth International Joint Conference on Natural Language Processing, IJCNLP 2011, Chiang Mai, Thailand, November 8-13, 2011*, pages 255–263. The Association for Computer Linguistics, 2011.

[127] Dora Melo, Irene Pimenta Rodrigues, and Vítor Beires Nogueira. Cooperative Question Answering for the Semantic Web. In Joaquim Filipe and Kecheng Liu, editors, *KMIS 2011 - Proceedings of the International Conference on Knowledge Management and Information Sharing, Paris, France, 26-29 October, 2011*, pages 258–263. SciTePress, 2011.

[128] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In Chiara Ghidini, Axel-Cyrille Ngonga Ngomo, Stefanie N. Lindstaedt, and Tassilo Pellegrini, editors, *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, ACM International Conference Proceeding Series, pages 1–8. ACM, 2011.

[129] Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-Value Memory Networks for Directly Reading Documents. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1400–1409. The Association for Computational Linguistics, 2016.

[130] David N. Milne and Ian H. Witten. Learning to link with wikipedia. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 509–518. ACM, 2008.

[131] Bonan Min, Marjorie Freedman, and Talya Meltzer. Probabilistic Inference for Cold Start Knowledge Base Population with Prior World Knowledge. In Mirella Lapata, Phil Blunsom, and Alexander Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 601–612. Association for Computational Linguistics, 2017.

[132] Alan F Murray and Peter J Edwards. Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on neural networks*, 5(5):792–802, 1994.

[133] Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. PATTY: A Taxonomy of Relational Patterns with Semantic Types. In Jun'ichi Tsujii, James Henderson, and Marius Pasca, editors, *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea*, pages 1135–1145. ACL, 2012.

[134] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. DyNet: The Dynamic Neural Network Toolkit. *CoRR*, abs/1701.03980, 2017.

[135] Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresová. SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing. In Daniel M. Cer, David Jurgens, Preslav Nakov, and Torsten Zesch, editors, *Proceedings of the 9th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2015, Denver, Colorado, USA, June 4-5, 2015*, pages 915–926. The Association for Computer Linguistics, 2015.

[136] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks. *CoRR*, abs/1511.08458, 2015.

[137] 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 Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pages 48–53. Association for Computational Linguistics, 2019.

[138] Xiaoman Pan, Taylor Cassidy, Ulf Hermjakob, Heng Ji, and Kevin Knight. Unsupervised Entity Linking with Abstract Meaning Representation. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1130–1139. The Association for Computational Linguistics, 2015.

[139] 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, July 6-12, 2002, Philadelphia, PA, USA*, pages 311–318. ACL, 2002.

[140] Alexandre Passos, Vineet Kumar, and Andrew McCallum. Lexicon Infused Phrase Embeddings for Named Entity Resolution. In Roser Morante and Wen-tau Yih, editors, *Proceedings of the Eighteenth Conference on Computational Natural Language Learning, CoNLL 2014, Baltimore, Maryland, USA, June 26-27, 2014*, pages 78–86. ACL, 2014.

[141] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global Vectors for Word Representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014.

[142] David Peterson, Shudi Gao, Ashok Malhotra, C. M. Sperberg-McQueen, and Henry S. Thompson. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. W3C Recommendation, April 2012. `https://www.w3.org/TR/xmlschema11-2/`.

[143] David C Plaut et al. Experiments on Learning by Back Propagation. 1986.

[144] Julien Plu, Giuseppe Rizzo, and Raphaël Troncy. Enhancing Entity Linking by Combining NER Models. In Harald Sack, Stefan Dietze, Anna Tordai, and Christoph Lange, editors, *Semantic Web Challenges - Third SemWebEval Challenge at ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*, volume 641 of *Communications in Computer and Information Science*, pages 17–32. Springer, 2016.

[145] Axel Polleres, Aidan Hogan, Andreas Harth, and Stefan Decker. Can we ever catch up with the Web? *Semantic Web*, 1(1-2):45–52, 2010.

[146] Jay M. Ponte and W. Bruce Croft. A Language Modeling Approach to Information Retrieval. In W. Bruce Croft, Alistair Moffat, C. J. van Rijsbergen, Ross Wilkinson, and Justin Zobel, editors, *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 275–281. ACM, 1998.

[147] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008. `https://www.w3.org/TR/rdf-sparql-query/`.

[148] Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. Stepwise Reasoning for Multi-Relation Question Answering over Knowledge Graph with Weak Supervision. In James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang, editors, *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 474–482. ACM, 2020.

[149] Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract Syntax Networks for Code Generation and Semantic Parsing. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1139–1149. Association for Computational Linguistics, 2017.

[150] Delip Rao, Paul McNamee, and Mark Dredze. Entity Linking: Finding Extracted Entities in a Knowledge Base. In Thierry Poibeau, Horacio Saggion, Jakub Piskorski,

and Roman Yangarber, editors, *Multi-source, Multilingual Information Extraction and Summarization*, Theory and Applications of Natural Language Processing, pages 93–115. Springer, 2013.

[151] Lev-Arie Ratinov and Dan Roth. Design Challenges and Misconceptions in Named Entity Recognition. In Suzanne Stevenson and Xavier Carreras, editors, *Proceedings of the Thirteenth Conference on Computational Natural Language Learning, CoNLL 2009, Boulder, Colorado, USA, June 4-5, 2009*, pages 147–155. ACL, 2009.

[152] Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale Semantic Parsing without Question-Answer Pairs. *Trans. Assoc. Comput. Linguistics*, 2:377–392, 2014.

[153] Giuseppe Rizzo, Amparo Elizabeth Cano Basave, Bianca Pereira, and Andrea Varga. Making Sense of Microposts (#Microposts2015) Named Entity rEcognition and Linking (NEEL) Challenge. In Matthew Rowe, Milan Stankovic, and Aba-Sah Dadzie, editors, *Proceedings of the the 5th Workshop on Making Sense of Microposts co-located with the 24th International World Wide Web Conference (WWW 2015), Florence, Italy, May 18th, 2015*, volume 1395 of *CEUR Workshop Proceedings*, pages 44–53. CEUR-WS.org, 2015.

[154] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[155] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[156] Gerard Salton, A. Wong, and Chung-Shu Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11):613–620, 1975.

[157] Guus Schreiber and Yves Raimond. RDF 1.1 Primer. W3C Working Group Note, June 2014. `https://www.w3.org/TR/rdf11-primer/`.

[158] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, 45(11):2673–2681, 1997.

[159] Yelong Shen, Jun Yan, Shuicheng Yan, Lei Ji, Ning Liu, and Zheng Chen. Sparse hidden-dynamics conditional random fields for user intent understanding. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 7–16. ACM, 2011.

[160] Kiyoaki Shirai, Kentaro Inui, Hozumi Tanaka, and Takenobu Tokunaga. An empirical

study on statistical disambiguation of japanese dependency structures using a lexically sensitive language model. In *Proceedings of Natural Language Pacific-Rim Symposium*, pages 215–220, 1997.

[161] He Shizhu, Zhang Yuanzhe, Liu Kang, Zhao Jun, et al. CASIA@ V2: A MLN-based Question Answering system over Linked Data. 2014.

[162] Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, Andre Valdestilhas, Diego Esteves, and Ciro Baron Neto. SPARQL as a Foreign Language. In Javier D. Fernández and Sebastian Hellmann, editors, *Proceedings of the Posters and Demos Track of the 13th International Conference on Semantic Systems - SEMANTiCS2017 co-located with the 13th International Conference on Semantic Systems (SEMANTiCS 2017), Amsterdam, The Netherlands, September 11-14, 2017*, volume 2044 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[163] Tommaso Soru, Edgard Marx, André Valdestilhas, Diego Esteves, Diego Moussallem, and Gustavo Publio. Neural Machine Translation for Query Construction and Composition. *CoRR*, abs/1806.10478, 2018.

[164] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. JSON-LD 1.0 – A JSON-based Serialization for Linked Data. W3C Recommendation, January 2014. `https://www.w3.org/TR/json-ld/`.

[165] Mark Steedman. A very short introduction to CCG. *Unpublished paper. http://www. coqsci. ed. ac. uk/steedman/paper. html*, 1996.

[166] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Large Ontology from Wikipedia and WordNet. *J. Web Sem.*, 6(3):203–217, 2008.

[167] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. Weakly Supervised Memory Networks. *CoRR*, abs/1503.08895, 2015.

[168] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W. Cohen. Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4231–4242. Association for Computational Linguistics, 2018.

[169] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

[170] Bilyana Taneva, M Kacimi El Hassani, and Gerhard Weikum. Finding images of rare and ambiguous entities. 2011.

[171] Stefan Thater, Hagen Fürstenau, and Manfred Pinkal. Contextualizing Semantic Representations Using Syntactically Enriched Vector Models. In Jan Hajic, Sandra Carberry, and Stephen Clark, editors, *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*, pages 948–957. The Association for Computer Linguistics, 2010.

[172] Trias Thireou and Martin Reczko. Bidirectional Long Short-Term Memory Networks for Predicting the Subcellular Localization of Eukaryotic Proteins. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 4(3):441–446, 2007.

[173] Cynthia A. Thompson and Raymond J. Mooney. Acquiring Word-Meaning Mappings for Natural Language Interfaces. *J. Artif. Intell. Res.*, 18:1–44, 2003.

[174] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs. In Claudia d'Amato, Miriam Fernández, Valentina A. M. Tamma, Freddy Lécué, Philippe Cudré-Mauroux, Juan F. Sequeda, Christoph Lange, and Jeff Heflin, editors, *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*, volume 10588 of *Lecture Notes in Computer Science*, pages 210–218. Springer, 2017.

[175] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016.

[176] Christina Unger and Philipp Cimiano. Pythia: Compositional Meaning Construction for Ontology-Based Question Answering on the Semantic Web. In Rafael Muñoz, Andrés Montoyo, and Elisabeth Métais, editors, *Natural Language Processing and Information Systems - 16th International Conference on Applications of Natural Language to Information Systems, NLDB 2011, Alicante, Spain, June 28-30, 2011. Proceedings*, volume 6716 of *Lecture Notes in Computer Science*, pages 153–160. Springer, 2011.

[177] Christina Unger, André Freitas, and Philipp Cimiano. An Introduction to Question Answering over Linked Data. In Manolis Koubarakis, Giorgos B. Stamou, Giorgos Stoilos, Ian Horrocks, Phokion G. Kolaitis, Georg Lausen, and Gerhard Weikum, editors, *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, volume 8714 of *Lecture Notes in Computer Science*, pages 100–140. Springer, 2014.

[178] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 7th Open Challenge on Question Answering over Linked Data (QALD-7). In Mauro Dragoni, Monika Solanki, and Eva Blomqvist, editors, *Semantic Web Challenges - 4th SemWebEval Challenge at ESWC 2017, Portoroz, Slovenia, May 28 - June 1, 2017, Revised Selected Papers*, volume 769 of *Communications in Computer and Information Science*, pages 59–69. Springer, 2017.

[179] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1747–1756. JMLR.org, 2016.

[180] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

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

[182] Denny Vrandečić. The Rise of Wikidata. *IEEE Intelligent Systems*, 28(4):90–95, 2013.

[183] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

[184] Claudia Wagner, Eduardo Graells-Garrido, David Garcia, and Filippo Menczer. Women through the glass ceiling: gender asymmetries in Wikipedia. *EPJ Data Science*, 5(1):5, Mar 2016.

[185] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[186] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[187] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433, 1995.

[188] William A. Woods. Progress in natural language understanding: an application to lunar geology. In *American Federation of Information Processing Societies: 1973 National Computer Conference, 4-8 June 1973, New York, NY, USA*, volume 42 of *AFIPS*

*Conference Proceedings*, pages 441–450. AFIPS Press/ACM, 1973.

[189] Gong-Qing Wu, Ying He, and Xuegang Hu. Entity Linking: An Issue to Extract Corresponding Entity With Knowledge Base. *IEEE Access*, 6:6220–6231, 2018.

[190] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144, 2016.

[191] Kun Xu, Yansong Feng, and Dongyan Zhao. Answering Natural Language Questions via Phrasal Semantic Parsing. In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014*, volume 1180 of *CEUR Workshop Proceedings*, pages 1260–1274. CEUR-WS.org, 2014.

[192] Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[193] Kun Xu, Lingfei Wu, Zhiguo Wang, Mo Yu, Liwei Chen, and Vadim Sheinin. Exploiting Rich Syntactic Information for Semantic Parsing with Graph-to-Sequence Model. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 918–924. Association for Computational Linguistics, 2018.

[194] Xuchen Yao and Benjamin Van Durme. Information Extraction over Structured Data: Question Answering with Freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 956–966. The Association for Computer Linguistics, 2014.

[195] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Bei-*

*jing, China, Volume 1: Long Papers*, pages 1321–1331. The Association for Computer Linguistics, 2015.

[196] Pengcheng Yin and Graham Neubig. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 7–12. Association for Computational Linguistics, 2018.

[197] Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural Machine Translating from Natural Language to SPARQL. *CoRR*, abs/1906.09302, 2019.

[198] Mohamed Amir Yosef, Johannes Hoffart, Ilaria Bordino, Marc Spaniol, and Gerhard Weikum. AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. *Proc. VLDB Endow.*, 4(12):1450–1453, 2011.

[199] Eman M. G. Younis, Christopher B. Jones, Vlad Tanasescu, and Alia I. Abdelmoty. Hybrid Geo-spatial Query Methods on the Semantic Web with a Spatially-Enhanced Index of DBpedia. In Ningchuan Xiao, Mei-Po Kwan, Michael F. Goodchild, and Shashi Shekhar, editors, *Geographic Information Science - 7th International Conference, GIScience 2012, Columbus, OH, USA, September 18-21, 2012. Proceedings*, volume 7478 of *Lecture Notes in Computer Science*, pages 340–353. Springer, 2012.

[200] Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cícero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. Improved Neural Relation Detection for Knowledge Base Question Answering. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 571–581. Association for Computational Linguistics, 2017.

[201] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *CoRR*, abs/1409.2329, 2014.

[202] John M. Zelle and Raymond J. Mooney. Learning to Parse Database Queries Using Inductive Logic Programming. In William J. Clancey and Daniel S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, pages 1050–1055. AAAI Press / The MIT Press, 1996.

[203] Wei Zhang, Jian Su, Chew Lim Tan, and Wenting Wang. Entity Linking Leveraging Automatically Generated Annotation. In Chu-Ren Huang and Dan Jurafsky, editors, *COLING 2010, 23rd International Conference on Computational Linguistics, Procee-*

*dings of the Conference, 23-27 August 2010, Beijing, China*, pages 1290–1298. Tsinghua University Press, 2010.

[204] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.00103, 2017.

[205] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over RDF: a graph data driven approach. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 313–324. ACM, 2014.

# Appendix A

# Neural Networks overview

We provide an overview of the main concepts and techniques relating to Neural Networks as are important for this work. Section A.1 and A.2 are based on Graves [72] work description for Supervised Sequence Labeling with Recurrent Networks and Section A.3 is based on [136].

## A.1. Neural Networks Fundamentals

Artificial Neural Networks (ANNs), or simply Neural Networks (NNs), are mathematical models inspired by how biological brains process information. Their basic structure is a network of small processing units, also seen as nodes, joined to each other by weighted connections. Similar to how synapses work in biological neurons, the network is activated by providing an input to some or all nodes, which spreads this activation throughout the rest of the network along the weighted connections.

### A.1.1. Multilayer Perceptrons

Among the different varieties of neural networks, **Feedforward Neural Networks** are structured in an acyclic way, meaning their connections do not form a cycle. One example are the multilayer perceptrons (MLP), which are arranged in layers with connections feeding forward from one layer to the next. An illustration can be seen in Figure A.1, where input data are passed to an **input layer** and then propagated through one or more **hidden layers** to the final **output layer**. This kind of architecture is more suitable for classification or function approximation tasks [72].

The process to pass data through layers is known as the **forward pass** of the network. Given an input vector of length $I$, an input layer of the same length would receive this input vector where each unit in the input layer calculates a weighted sum. For a hidden unit $h$, we refer to this sum as the network input to unit $h$, denoted as $a_h$. Denoting $w_{\mathrm{i}j}$ as the weight

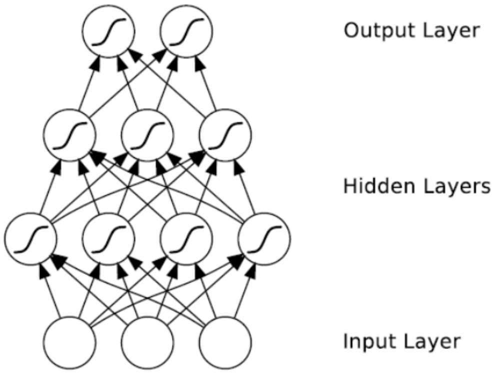


Figure A.1: Multilayer perceptron [72].

from unit i to unit $j$, the formula of $a_h$ for a layer $H_l$ is calculated using the formula in Equation A.1.

$$a_h = \sum_{h' \in H_{l-1}}^{I} w_{h'h}\, b_{h'} \tag{A.1}$$

where $b_{h'}$ corresponds to the final activation function of the previous layer. The $b_h$ value is calculated by applying an activation function $\theta_h$ over $a_h$, as seen in Equation A.2. Note that for the first hidden layer, the previous layer is the input layer.

$$b_h = \theta_h(a_h) \tag{A.2}$$

This activation function $h$ can vary, though some of the most common functions used are the hyperbolic tangent A.3 or sigmoid A.4 functions. Two important features about activation functions: they are non-linear and differentiable. Non-linearity allows the network to build more complex internal features (e.g. build more flexible boundaries in a classification task), and differentiability is required to perform the backward pass that will be mentioned below.

$$tanh(x) = \frac{\mathrm{e}^{2x} - 1}{\mathrm{e}^{2x} + 1} \tag{A.3}$$

$$sigmo\mathrm{id}(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{A.4}$$

This process of summation and activation is repeated for $L$ hidden layers until reaching the output layer, where the output vector $y$ is determined by using the activation of the last hidden layer $H_L$. Then, the network input $a_k$ to each output unit $k$ is calculated by summing over the units of the connected to it, the same way as is expressed in Equation A.1. The output activation function to be used depends on the task the network aims to fulfill. For simple binary classification, the sigmoid function A.4 is applied since its values between 0 or 1 can be seen as a binary probability $p(z|x)$, with $z$ being the target vector. Furthermore, if the classification task includes more than 2 classes, there is a convention to have $K$ output units, and normalize the output activations with the softmax function A.5. Therefore, the class probability for $C_k$ given the output $x$ is represented by Equation 5

$$\begin{aligned} p(C_k|x) &= y_k \\ &= softmax(a_k) \\ &= \frac{\mathrm{e}^{a_k}}{\sum_{k'=1}^{K} \mathrm{e}^{a_{k'}}} \end{aligned} \tag{A.5}$$

Lastly, a 1-to-K scheme is used to represent the target class $z$ where $z$ is represented as a one-hot vector (e.g. if $K = 4$, the class $C2$ is represented as $[0,\ 1,\ 0,\ 0]$). More formally, the way to express the target probabilities are as follows:

$$p(z|x) = \prod_{k=1}^{K} y_k^{z_k}$$

In the context of pattern classification, the class label that should be chosen corresponds to the most active output unit, i.e. the higher value from all target probabilities.

### A.1.2. Network Training

In order to have an idea whether a neural network is working as expected, a loss function is used. As per activation functions, the loss function to be used depends on the task the Neural Network is performing. For example, for multiple classification the maximum-likelihood function is commonly used as a loss function [21]:

$$L(y_k, z) = \sum_{k=1}^{K} z_k ln\ y_k$$

Neural networks are able to learn, i.e. they can generalize to unseen data, so they can be trained by minimizing the loss function $L$. One of the simplest algorithms to perform such training process is the *gradient descent* algorithm. Gradient descent consists of repeatedly taking a small, fixed-size step in the direction of the negative error gradient of the loss function, which can also be seen as going in the opposite direction of the negative slope of the loss function. Note that we perform gradient descent over a training dataset, while we save a test dataset that is not used to train but to evaluate the overall performance after training.

Thus, a weight update $\Delta w^n$, also known as gradient , is used to update the weight vector $w^n$ from the $n^{th}$ network layer. The gradient $\Delta w^n$ consists of the partial derivative of the loss function when the weight vector $w^n$ varies. This derivative is adjusted by a **learning rate** $\alpha \in [0, 1]$ which limits how quick the training process is converging. Then, on each gradient descent iteration i the weight vector $w^n$ is updated as follows:

$$w^n_{\mathrm{i}} = w^n_{\mathrm{i}-1} - \Delta w^n_{\mathrm{i}-1} = w^n_{\mathrm{i}-1} - \alpha \frac{\partial L}{\partial w^n_{\mathrm{i}-1}}$$

The backpropagation technique is commonly used to calculate these gradients [154, 187, 185], often referred to as the **backward pass** of the network. Backpropagation consists of the repeated application of chain rule for partial derivatives. For example, for a multiclass network, the application of the chain rule over the loss function defined as $\frac{\partial L(x,z)}{\partial w_{\mathrm{i}j}} = \frac{\partial L(x,z)}{\partial a_j} \frac{\partial a_j}{w_{\mathrm{i}j}}$, which then can be deduced by applying the chain rule again over the unknown gradients. Note that this process has to be performed over every weight of each layer on the network.

The training algorithm is repeated until a **stopping criteria** is met (e.g. stop after a fixed amount of steps, when reaching a certain loss threshold, or when failing to reduce the loss on a given number of consecutive steps). Usually, this process involves using the entire training data more than one time, where an entire pass over the data is known as one **epoch**. By the end of the training process, we expect to have the neural network's weights such that the loss function has reached a value as close as possible to the global minimum when evaluating over test examples (i.e. it can predict as best as possible over the training data).

The training process involves many issues that can lead to bad performance, a long time to train models, or divergence problems (i.e. training process never ends). One common issue is when the gradient descent process gets stuck in local minimums, which can be addressed by adding a momentum term to reduce learning inertia [143]. To boost training time, many variants of gradient descent have been proposed such as stochastic gradient descent or mini-batch gradient descent [106].

Another issue related with bad performance is **overfitting**, which causes the network not to be able to generalise properly since it *"memorizes"* the data from the training dataset. One way to see if a model is overfitted is to check the loss function values evolution over the training process for the training and the test set: if the loss for the test is not decreasing but instead increasing while the loss for the training set is constantly decreasing, the model is getting overfitted.

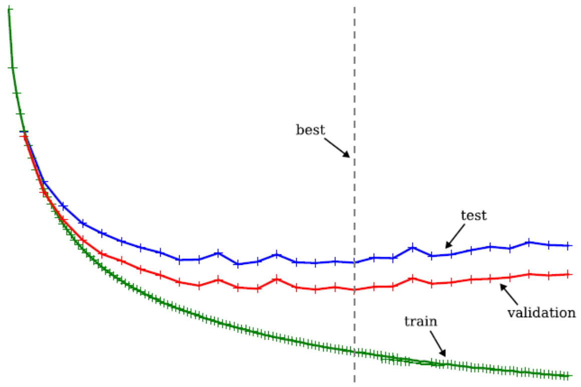


Figure A.2: Example of early stopping analysis using validation data [72].

One solution that helps to address the overfitting issue is to use a small portion of the training set as a validation set to include an **early stopping criteria**. This validation set is not used to train the network but to perform validation steps every certain amount of training steps. Then, the loss values for the validation steps are used to decide when to stop training. For example, Figure A.2 shows the losses over all three data sets (train, validation, test) through the training process. Then, we can detect that the best weight values are found just before the validation loss stops its decreasing tendency and starts increasing again (where the “best” stripped vertical line is placed). There are other techniques to reduce overfitting known as regularizers based on input noise [7, 102, 21] or weight noise [132, 96].

Though most of the performance of ANNs relies on learnable parameters (such as layers’ weights), there are other parameters that can be set manually to improve the model’s performance, also known as **hyperparameters**. Some hyperparameters could be the number of hidden layers, the number of hidden units per layer, the learning rate, among others.

Lastly, it is also important to understand how the network input is represented. When we mention the **input representation**, we refer to the representation of the information required to predict the outputs, such as the input vector or the network weights. One procedure is input standardisation, which consists of normalizing the components of the input vectors to have mean 0 and standard deviation 1 over the training set. This standardization does not alter the information but helps to improve performance by limiting the values of the input vector to a more suitable range for a standard activation function [106]. Note that the validation set and test set have to be standardised using the same distribution used for the training set.

Another procedure is **weight initialisation**, which helps gradient descent to “break symmetry” between units [106] and avoid training divergence. Weight initialisation then is to initialise weights with either a random distribution in the range of small values or a Gaussian distribution with mean 0 and standard deviation 0.1.

After reviewing the fundamental concepts needed to understand how neural networks are structured and trained, we will review two neural networks architectures used in this work: Recurrent Neural Networks and Convolutional Neural Networks.

## A.2. Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a generalization of traditional feedforward Neural Networks that allows cyclical connections [72]. While MLPs can only map from input to output vectors, RNNs can map the entire history of previous inputs to an output. Hence, RNN connections allow the network to have "memory" of previous inputs, thus influencing the network output. Furthermore, RNNs are more fit for tasks involving sequence data, such as text or audio. An example of a RNN architecture can be seen in Figure A.3.

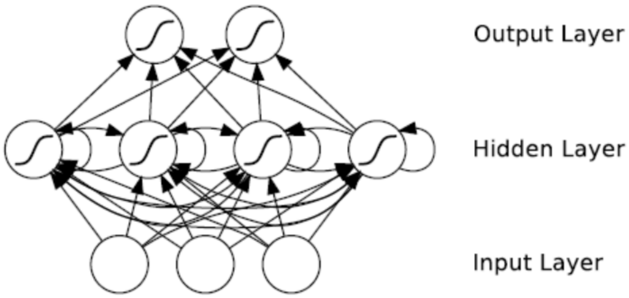


Figure A.3: Recurrent Neural Network architecture example [72] .

A standard RNN computes the **forward pass** the same way as an MLP with a single hidden layer, with the difference that the activations arrive at the hidden layer from both the current external input and the hidden layer activations from the previous time step. Given a sequence of inputs $(x_1, \ldots, x_T)$, with $T$ the input length, the forward pass for an RNN with $I$ input units and $H$ hidden units is computed using the following formula:

$$a_h^t = \sum_{\mathrm{i}=1}^{I} w_{\mathrm{i}h}\, x_{\mathrm{i}}^t + \sum_{h'=1}^{H} w_{h'h}\, b_{h'}^{t-1} \tag{A.6}$$

where $x_{\mathrm{i}}^t$ denotes the value of input i, $a_j^t$ is the network input to unit $j$, and $b_j^t$ is the activation unit of unit $j$, all three at time t. Then, the non-linearity $h$ is applied the same way as for an MLP, where functions such as the sigmoid function or softmax function are again a common choice:

$$b_h^t = \theta_h(a_h^t) \tag{A.7}$$

The entire learning process for RNNs is then summarized as a recursive application of the Equations A.6 and A.7, starting at $t = 1$. Since initial values $b_{\mathrm{i}}^0$ are needed, they can be initialized using the same weight initialization methods mentioned above. The output units $a_k$ can be calculated at the same way as the hidden activations:

$$a_k^t = \sum_{h=1}^{H} w_{hk}\, b_h^t \tag{A.8}$$

Then, the final activation function also depends on the task involved. For connection temporal classification (CTC) tasks, such as sequence labeling or translation, it is common to use the softmax function [74]. The CTC loss function is used, that is defined as the negative log probability of correctly labelling all examples in the training set.

The **backward pass** can be performed using a backpropagation through time (BPTT) algorithm [187, 186], which is an algorithm based on the standard backpropagation process but adapted to RNNs. The BPTT algorithm also consists of a repeated application of the chain rule, with the difference that, aside from the output layer, the loss function also depends on the activation of the hidden layer through its influence on the hidden layer at the next timestep. The derivatives can be calculated using the same procedure described in the *Network Training* subsectionA.1.2, but taking into consideration that the same weights are being reused at every timestep.

Since the classical RNN architecture only looks to past information, the **Bidirectional Recurrent Neural Network** (BRNN) was proposed to also include future context [158]. The BRNN's structure consists of two separate recurrent hidden layers, both connected to the same output layer. This idea allows a forwards and backwards training sequence, where the layer that performs the backward training receives the input sequence in the opposite direction. The output layer is not updated until both hidden layers have processed the entire input sequence.

### A.2.1. Long Short-Term Memory

Though RNNs work well with short sentences, their performance decreases with long sentences that involve long term dependencies due to the **vanishing gradient** problem [87, 86], which occurs when the influence of the given input on the hidden layer (and therefore on the network output), either decays or explodes exponentially through the recurrent connections. In order to address this issue, the **Long Short-Term Memory** (LSTM) model [87] was proposed. The LSTM model allows models to perform well on tasks which require long range temporal dependencies, such as Sequence Labeling [93, 117], Machine Translation [190] or Summarization [119]. As per RNNs, the LSTM also has a bidirectional variant, also known as BiLSTM [76, 32, 172].

An LSTM network is similar to a standard RNN, except that the summations units in the hidden layer are replaced by **memory blocks**, as shown in Figure A.4. This structure allows the memory cells to store and access information over long periods of time, thereby reducing the effects of the vanishing gradient problem.

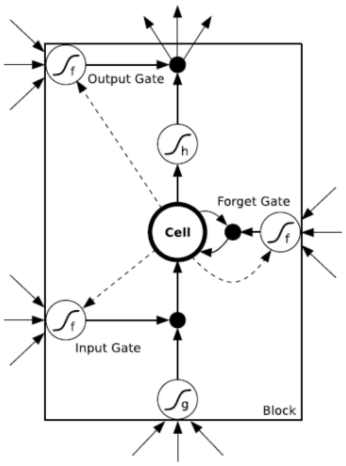


Figure A.4: LSTM memory block with one cell [72].

The following equations presented below are the formulas used to perform the forward pass evaluation over an LSTM with a single memory block for a timestep $t$. For multiple blocks the computations are repeated for each block. The values of $w_{\mathrm{i}j}$, $a_j^t$ and $b_j^t$ have the same definition used before.

First, the **Input Gates**, denoted as $a_\iota^t$ A.9 and $b_\iota^t$ A.10. The number of inputs is denoted by $I$, the number of cells in the hidden layer is $H$ and the number of memory cells is $C$. The gate activation function $f$ most commonly used is the sigmoid function, so the gate activations are between 0 (gate closed) and 1 (gate open). The weight $w_{c\iota}$ represents the peephole weight from cell $c$ to the input gate. The state of the cell $c$ at time $t$ is denoted as $s_c^t$, which is calculated using Equation A.14.

$$a_\iota^t = \sum_{\mathrm{i}=1}^{I} w_{\mathrm{i}\iota}\, x_{\mathrm{i}}^t + \sum_{h=1}^{H} w_{h\iota}\, b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota}\, s_c^{t-1} \tag{A.9}$$

$$b_\iota^t = f(a_\iota^t) \tag{A.10}$$

Then, the **Forget Gates**, denoted as $a_\phi^t$ A.11 and $b_\phi^t$ A.12. The weight $w_{c\phi}$ represents the peephole weight from cell $c$ to the forget gate. Besides that, other symbols are equivalent to those mentioned for the input gate.

$$a_{\phi}^{t} = \sum_{\text{i}=1}^{I} w_{\text{i}\phi}\, x_{\text{i}}^{t} + \sum_{h=1}^{H} w_{h\phi}\, b_{h}^{t-1} + \sum_{c=1}^{C} w_{c\phi}\, s_{c}^{t-1} \tag{A.11}$$

$$b_{\phi}^{t} = f(a_{\phi}^{t}) \tag{A.12}$$

The **Cells**, denoted as $a_c^t$ A.13 and $s_c^t$ A.14. The cell input activation function $g$ is usually hyperbolic tangent or sigmoid.

$$a_{c}^{t} = \sum_{\text{i}=1}^{I} w_{\text{i}c}\, x_{\text{i}}^{t} + \sum_{h=1}^{H} w_{hc}\, b_{h}^{t-1} \tag{A.13}$$

$$s_{c}^{t} = b_{\phi}^{t}\, s_{c}^{t-1} + b_{\iota}^{t}\, g(a_{c}^{t}) \tag{A.14}$$

Next, the **Outputs Gates**, denoted as $a_\omega^t$ A.15 and $b_\omega^t$ A.16. The weight $w_{c\omega}$ represent the peephole w eight from cell $c$ to the output gate.

$$a_{\omega}^{t} = \sum_{\text{i}=1}^{I} w_{\text{i}\omega}\, x_{\text{i}}^{t} + \sum_{h=1}^{H} w_{h\omega}\, b_{h}^{t-1} + \sum_{c=1}^{C} w_{c\omega}\, s_{c}^{t-1} \tag{A.15}$$

$$b_{\omega}^{t} = f(a_{\omega}^{t}) \tag{A.16}$$

Finally, the **Cell Outputs**, denoted as $b_c^t$ A.17. The cells outputs $b_c^t$ are the only ones connected to the other blocks in the layer. The index h is used to refer to cell outputs from other blocks in the hidden layer, if they exist. As per $g$, the output activation function $h$ is usually hyperbolic tangent or sigmoid, though sometimes the identity function can be used.

$$b_{c}^{t} = b_{\omega}^{t}\, f(s_{c}^{t}) \tag{A.17}$$

## A.3. Convolutional Neural Networks

The creation of **Convolutional Neural Networks** (CNNs) responds to the need to process certain types of data: images [136]. Traditional ANNs do not perform well when processing images since its architecture does not properly support the computational complexity that involves processing large images as input. Whereas a 32×32 image will be no problem to a traditional ANN, since it will require only 1024 parameters for a single neuron, images tend to have more resolution. On a higher scale, an image of 1024×1024 will instead require 1,048,576 parameters, which is a substantial increase. Moreover, if we consider colored images (RGB), a 1024×1024 RGB image would require 3,145,728 parameters. There is then a noticeable drawback of using standard feed-forward models where nodes are often connected to each node from the previous layer.

Convolutional Neural Networks share many similarities with standard ANNs in the way that both are composed of a set of neurons that are capable of learning, where each neuron receives an input and performs many operations, which commonly is a scalar product followed by an activation function. The difference resides in that CNNs are based on the idea that the input is shaped as an image. This idea allows the CNN architecture to adapt to this specific type of data. Then, a CNN architecture is built using three different types of layers: convolutional layers, pooling layers and fully-connected layers (same layers used in traditional ANNs). Additionally, each layer is organised into three dimensions: the spatial dimension (width and height) and the number of channels (also known as depth, which is not the same as the number of layers). An example of a CNN architecture for pattern image classification is shown in Figure A.5.

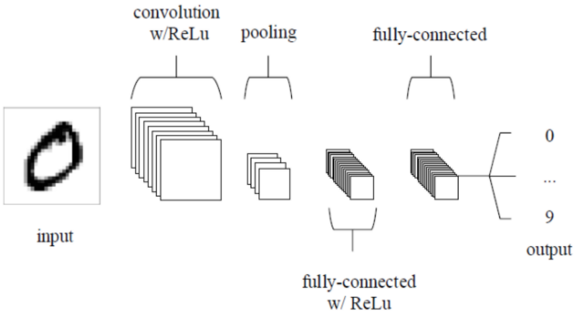


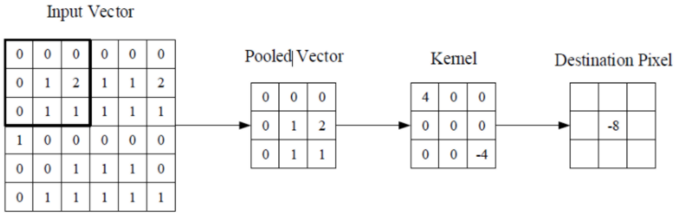
Figure A.5: Convolutional Neural Network for pattern image classification [136].

### A.3.1. Convolutional layer

A **convolution layer** is the central component of CNNs, which determines the output of neurons using calculations based on local regions of the input. This type of layer is based on learnable kernels, which define local convolution operations over the input vector. A convolution consists of the scalar product for each value in a kernel of dimension M×N over a local region with the same size as the kernel used:

$$(X * w)_{\mathrm{i},j} = \sum_{m=1}^{M}\sum_{n=1}^{N} X_{m,n} \cdot w_{\mathrm{i}-m,j-n} \tag{A.18}$$

For example, in Figure A.6 the convolution is being applied over a 3×3 pooled vector, which is the size of the kernel $w$, that is, the local region of the entire input vector $X$. Though these kernels usually have a small spatial dimensionality, they are spread along the whole input vector. Besides kernel dimension, an application of padding over the input vector is also possible, which is the process of padding the border of the input with zeros. **Padding** controls both the dimensionality of the output volumes and gives more relevance to the input borders. Lastly, the **stride** is the amount of spaces the kernel is moved between each convolution. By setting a stride greater than 1, it is possible to reduce the amount of overlap and thus reduce the dimension of the activation output.



| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 1 | 2 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 2 |
| 0 | 1 | 1 |

| | | |
|---|---|---|
| 4 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | -4 |

| | | |
|---|---|---|
| | | |
| | -8 | |
| | | |

Figure A.6: Convolution operation example [136].

Let $N$ be the size of a input vector of size N×N, $F$ the kernel F×F dimension, $P$ the padding size, and $S$ the stride value; the final dimension of the output volume will be $\left\lfloor \frac{N-F+2P}{S} + 1 \right\rfloor$. Note that, in the same way an image can be represented in 3 dimensions, the kernel can be extended to a third dimension by increasing the **number of channels**, thus giving control over the output depth of the convolutional layer. The number of channels, stride and padding are hyperparameters that can be optimized.

Kernel values are the trainable parameters which a CNN can tune through the same type of learning process ANNs perform. The composition of convolutional layers allows to reduce the dimensionality of a Neural Network in terms of learnable parameters, based on the assumption of parameter sharing. This assumption says that *"if one region is useful to compute at a set spatial region, then it is likely to be useful in another region"*. The constraints of each activation within the output volume to the same weights and bias means a significant decrease in the number of parameters used in a convolution layer. Then, the backward pass for each neuron in the output represents the overall gradient across channels, where only a single set of weights is updated.

After the application of the convolution, an activation function is applied over the output volume. The most common choice is the rectified linear unit (ReLu) which is an elementwise function that given a certain threshold $\alpha$ will set all values less than $\alpha$ to 0 .

## A.3.2. Pooling layer

A pooling layer aims to reduce the volume of an input representation with the purposes of reducing the computational complexity of the model. After each activation from a convolutional layer, a pooling layer can be applied to scale its dimensionality through a reducing function. The most common type of pooling is the max-pooling layer, which is a kernel that applies a MAX reduction on a local region as per a convolutional kernel. Other examples are average pooling, or general pooling that applies average reduction and $L_1/L_2$ normalization respectively.

Though pooling layers also include settings such as kernel size, stride or number of channels, they do not add learnable parameters. However, they do influence the backward pass to calculate the gradients. Lastly, it is recommended to keep a low stride and kernel size since its application could negatively affect performance if large values are used.

### A.3.3. Common architectures

As mentioned before, a CNN architecture is commonly built with an input layer, which receives the values of the image, followed by various stacked convolutional layers, each one followed by pooling layers, and a final stack of fully-connected layers. However, defining exactly the amount of layers to use is not a simple task. In fact, most of the literature is based on standard architectures that have shown good results on certain image processing tasks.

Among the most popular architectures, ImageNet [103] is a Deep Convolutional Network with five convolutional layers, some followed by max-pooling layers, followed by two fully-connected layers. Another example is ResNet [83], which includes **residual connections** that aim to reduce the vanishing gradient problem that a very dense convolutional network can suffer. The main principle of residual connections is to create connections between non-adjacent layers that skip a certain amount of layers.

# Appendix B

# Question Answering Dataset

We provide more details on the data used for the training and validation of our system.

## B.1. Normalized Dataset Format

As mentioned in the *Experimental Design* chapter 4, we proposed a dataset format so the process of generating the Query Template dataset and the Sequence Labeling datasets could be simplified. This normalized format is also used for the other test datasets (`QALD-7` and `WikiSPARQL`). In Listing B.1 we can see an example of what information each case contains:

```
    "question_id": 30226,
    "natural_language_question": "Did Alexander Hamilton practice law?",
    "query_answer": [
        {
            "query_id": 0,
            "sparql_query":
                "ASK WHERE { wd:Q178903 wdt:P106 wd:Q40348 }",
            "entities": [
                {"label": "Alexander Hamilton", "entity": "wd:Q178903"},
                {"label": "lawyer", "entity": "wd:Q40348"}
            ],
            "slots": [
                {"slot": "<sbj_1>", "label": "Alexander Hamilton"},
                {"slot": "<obj_1>", "label": "lawyer"}
            ],
            "sparql_template": "ASK WHERE { <sbj_1> wdt:P106 <obj_1> }"
        }
    ]
```

Listing B.1: Example of one `LC-QuAD 2` case following our proposed normalized format.

- **Question ID**: the unique identifier of the question; for `LC-QuAD 2` we decided to treat each verbalized and paraphrased version of the normalized question as a separate case. Verbalized cases are then numbered from $0$ to $30,225$, and paraphrased cases from $30,226$ to $60,451$.

- **Natural Language Question**: the question text (verbalized or paraphrased version).

- **Query Answer**: a list of possible `SPARQL` query valid responses. We allow more than one possible query since for each question there are many ways to reach the expected answer (for example, *"What is the biggest country?"* may refer to population, area, etc.; however in this work we only have one `SPARQL` query per case). Each query answer case has the following fields:

  – **Query ID**: unique identifier to identify query answers for the same question.

  – **SPARQL query**: the `SPARQL` query string.

  – **Entities**: list of expected annotations of the entities being used in the `SPARQL` query answer. Each annotation contains the entity URL and the label of the question associated with that entity.

  – **Slots**: list of expected slots to use for the Slot Filling system. Each slot contains the associated label in the question and its corresponding placeholder in the Query Template.

  – **SPARQL Template**: the Query Template derived from the `SPARQL` query.

## B.2. LC-QuAD 2 base templates

In the *Results* chapter 5 we displayed an analysis based on the 22 base templates used for building the `LC-QuAD 2` dataset [47]. According to each base template structure, we established which cases are considered **complex cases**, which allows us to estimate the percentage of complex questions this dataset contains. A base template is considered a *complex case* if it includes any of the following operations: (1) counting, (2) filtering, (3) ordering, (4) use of strings or numbers,(5) use of property statements, or (6) returns more than one variable.

Next, we present a brief overview of each one of these base templates along with a representative example from the dataset:

1. **ask_one_fact**: ASK query with one query triple.

   - Example: *Did Alexander Hamilton practice law?*

- Query:

```
ASK WHERE {
    wd:Q178903 wdt:P106 wd:Q40348
}
```

2. **ask_one_fact_with_filter**: ASK query with one query triple and a numeric filter operation (either less than, equal, or greater than).

   - Example: *Does the standard molar entropy of silver equal 34.08?*

   - Query:

   ```
   ASK WHERE {
       wd:Q1090 wdt:P3071 ?obj filter(?obj = 34.08)
   }
   ```

3. **ask_two_facts**: ASK query with two query triples. Note that the same subject and property are used in both query triples.

   - Example: *Was William Henry Harrison both a United States senator and Governor of Indiana?*

   - Query:

   ```
   ASK WHERE {
       wd:Q11869 wdt:P39 wd:Q16147601 .
       wd:Q11869 wdt:P39 wd:Q13217683
   }
   ```

4. **count_one_fact_object**: SELECT query with COUNT operation from one query triple. Count the number of objects that match the query triple.

   - Example: *How many Latin conjugations are there?*

   - Query:

   ```
   SELECT (COUNT(?obj) AS ?value ) WHERE {
       wd:Q397 wdt:P5206 ?obj
   }
   ```

5. **count_one_fact_subject**: SELECT query with COUNT operation from one query triple. Count the number of subjects that match the query triple.

   - Example: *What is the number of spore print colors for olive?*

- Query:

```
SELECT (COUNT(?sbj) AS ?value ) WHERE {
    ?sbj wdt:P787 wd:Q864152
}
```

6. **rank_instance_of_type_one_fact**: SELECT query with two query triples with sorting and limit. The first query triple always uses the property instance of (wdt:P31). Ordering might vary (ascending or descending).

   - Example: *What battery power station has the highest amount of energy storage capacity?*

   - Query:

```
SELECT ?ent WHERE {
    ?ent wdt:P31 wd:Q810924 .
    ?ent wdt:P4140 ?obj
} ORDER BY DESC(?obj) LIMIT 5
```

7. **rank_max_instance_of_type_two_facts**: SELECT query with three query triples using sorting and limit. The first query triple always uses the property instance of (wdt:P31). Descending ordering is used to get the maximum value.

   - Example: *What is the largest village in Muchinigi Puttu?*

   - Query:

```
SELECT ?ent WHERE {
    ?ent wdt:P31 wd:Q532 .
    ?ent wdt:P2046 ?obj .
    ?ent wdt:P131 wd:Q11107378
} ORDER BY DESC(?obj) LIMIT 5
```

8. **rank_min_instance_of_type_two_facts**: Same as the previous base template but with ascending ordering to get the minimum value.

   - Example: *What is the name of a manned spacecraft in low Earth orbit with smallest periapsis?*

   - Query:

```
SELECT ?ent WHERE {
    ?ent wdt:P31 wd:Q7217761 .
    ?ent wdt:P2244 ?obj .
    ?ent wdt:P522 wd:Q663611
} ORDER BY ASC(?obj) LIMIT 5
```

9. **select_object_instance_of_type**: SELECT query with two query triples. The first query triple always uses the property instance of (wdt:P31). Return the entities that are the object of the first query triple.

    - Example: *What is the prefecture of Hiroshima in Japan?*

    - Query:

```
SELECT DISTINCT ?obj WHERE {
    wd:Q34664 wdt:P131 ?obj .
    ?obj wdt:P31 wd:Q50337
}
```

10. **select_object_using_one_statement_property**: SELECT query with three query triples. Uses one property statement and one property qualifier.

    - Example: *When did Louis XVIII of France, husband of Marie Josephine of Savoy, die?*

    - Query:

```
SELECT ?value WHERE {
    wd:Q7750 p:P26 ?s .
    ?s ps:P26 wd:Q231844 .
    ?s pq:P582 ?value
}
```

11. **select_one_fact_object**: SELECT query with one query triple. Return the entities that are the subject of that query triple.

    - Example: *Which are the characters for La Malinche?*

    - Query:

```
SELECT DISTINCT ?answer WHERE {
    ?answer wdt:P674 wd:Q230314
}
```

12. **select_one_fact_subject**: SELECT query with one query triple. Return the enti-

ties that are the object of that query triple.

- Example: *Which is the electric charge for antihydrogen?*
- Query:

```
SELECT DISTINCT ?answer WHERE {
    wd:Q216121 wdt:P2200 ?answer
}
```

13. **select_one_qualifier_value_and_object_using_one_statement_property**: SELECT query with three query triples. Uses one property statement and one property qualifier. Only one entity used for the first query triple.

    - Example: *What grant did Konrad Lorenz win, and who won it with him?*
    - Query:

```
SELECT ?value1 ?obj WHERE {
    wd:Q78496 p:P166 ?s .
    ?s ps:P166 ?obj .
    ?s pq:P1706 ?value1 .
}
```

14. **select_one_qualifier_value_using_one_statement_property**: SELECT query with three query triples. Uses one property statement and one property qualifier. Two entities used for building each query, where the second entity was used as the object of the property qualifier triple.

    - Example: *What role did Theodore Roosevelt occupy after William McKinley?*
    - Query:

```
SELECT ?obj WHERE {
    wd:Q33866 p:P39 ?s .
    ?s ps:P39 ?obj .
    ?s pq:P1365 wd:Q35041
}
```

15. **select_subject_instance_of_type**: SELECT query with two query triples. The first query triple always uses the property instance of (wdt:P31). Return the entities that are the subject of the first query triple.

    - Example: *Which irresistible illness is caused by Staphylococcus aureus?*

- Query:

```
SELECT DISTINCT ?sbj WHERE {
    ?sbj wdt:P828 wd:Q188121 .
    ?sbj wdt:P31 wd:Q18123741
}
```

16. **select_subject_instance_of_type_contains_word**: SELECT query with two query triples. The first query triple always uses the property instance of (wdt:P31). The second query triple also includes a filter over the string value to check whether the target word is contained in this string (includes filtering for the English language).

    - Example: *What is the title of a human that contains the word vitellius in their name.*

    - Query:

```
SELECT DISTINCT ?sbj ?sbj_label WHERE {
    ?sbj wdt:P31 wd:Q5 .
    ?sbj rdfs:label ?sbj_label .
    FILTER(CONTAINS(lcase(?sbj_label), 'vitellius')) .
    FILTER (lang(?sbj_label) = 'en')
} LIMIT 25
```

17. **select_subject_instance_of_type_starts_with**: SELECT query with two query triples. The first query triple always uses the property instance of (wdt:P31). The second query triples also includes a filtering over the string value to check whether this string starts with a target letter (including filtering for the English language).

    - Example: *What are the video games which start with the letter W?*

    - Query:

```
SELECT DISTINCT ?sbj ?sbj_label WHERE {
    ?sbj wdt:P31 wd:Q7058673 .
    ?sbj rdfs:label ?sbj_label .
    FILTER(STRSTARTS(lcase(?sbj_label), 'w')) .
    FILTER (lang(?sbj_label) = 'en')
} LIMIT 25
```

18. **select_two_answers**: SELECT query with two query triples. Return two variables (also known as double intention), with each one being the object of each query triple.

    - Example: *Where was Jane Austen born and where did she die?*

- Query:

```
SELECT ?ans_1 ?ans_2 WHERE {
    wd:Q36322 wdt:P19 ?ans_1 .
    wd:Q36322 wdt:P20 ?ans_2
}
```

19. **select_two_facts_left_subject**: SELECT query with two query triples.

    - Example: *Who were the creators of The Late Awesome Planet Soil?*

    - Query:

    ```
    SELECT ?answer WHERE {
        wd:Q22081649 wdt:P144 ?obj .
        ?obj wdt:P50 ?answer
    }
    ```

20. **select_two_facts_right_subject**: SELECT query with two query triples. Similar to the previous template (in fact we could not find any difference between both cases).

    - Example: *What time zone is Arizona State University in?*

    - Query:

    ```
    SELECT ?answer WHERE {
        wd:Q670897 wdt:P17 ?obj .
        ?obj wdt:P421 ?answer
    }
    ```

21. **select_two_facts_subject_object**: SELECT query with two query triples. Uses one entity in the subject of the first query triple, and another one in the object of the second query triple.

    - Example: *What lake of Sao Jorge island has the tributary Curoca River?*

    - Query:

    ```
    SELECT ?answer WHERE {
        wd:Q743362 wdt:P206 ?answer .
        ?answer wdt:P974 wd:Q10361834
    }
    ```

22. **select_two_qualifier_values_using_one_statement_property**: SELECT query with four query triples. Uses one property statement and two property qualifiers. Re-

turn the two object values of each query triple using a property qualifier.

- Example: *Give me the year and name of the person with whom Bob Barker shared the MTV Movie Award for Best Fight.*

- Query:

```
SELECT ?value1 ?value2 WHERE {
    wd:Q381178 p:P166 ?s .
    ?s ps:P166 wd:Q734036 .
    ?s pq:P585 ?value1 .
    ?s pq:P1706 ?value2
}
```