Natural Language Question Answering agent over Knowledge Graphs

Aryan Veer Singh, B.Tech

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Prof. Gaye Stephens

August 2021

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

[illegible]

Aryan Veer Singh

August 19, 2022

Permission to Lend and/or Copy

[illegible]

Aryan Veer Singh

August 19, 2022

Natural Language Question Answering agent over Knowledge Graphs

Aryan Veer Singh, Master of Science in Computer Science
University of Dublin, Trinity College, 2021

Supervisor: Prof. Gaye Stephens

There is a vast amount of open data available on the Semantic Web. This data is structured, can be understood by machines as it has the semantics and can be queried using a query language. But this querying requires some expert knowledge of a language called SPARQL. Also after querying this data, the answers are also not easily readable by a lay user as it is in RDF triples format. So in this project, we have created such a system which takes a question in natural language as input, and returns the answer for that question in the same natural language. The inner workings of the system is kept abstract from the end user which makes it easy to use. Also we have used state-of-the art methods in order to implement this Question Answering system over Knowledge Bases.

Acknowledgments

Thank you Mom & Dad and Manpreet.

Also I would like to thank Prof. Declan O'Sullivan for teaching the module 'Knowledge and Data Engineering', after which I decided to work on following research topic. Also thanks to Prof. Gaye Stephen for being my supervisor and helping me in figuring things out for this research project.

Aryan Veer Singh

University of Dublin, Trinity College
August 2021

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There is an enormous amount of data and information that exists on the Semantic Web in the form of Knowledge Bases. The Semantic Web was developed by Tim Berners-Lee who, defined it as the following: "The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better-enabling computers and people to work in cooperation". The word Semantic here means *meaning*, which enables the practical use of the information available. As meaning is often not present in most information sources, so it requires human understanding or some complex programming to extract or even supply the meaning. Due to weak semantics, the search is limited to the exact matches. Semantics provide useful meaning to a keyword symbol, for example, keyword *building* could mean constructing something, or even a structure. So this ambiguity in the meaning of the keyword is removed by relations. *Building* relates to other keywords such as construction, building plans, architect, site, etc. - hence relationships uncover semantics. Along with this, the arrangement of the keywords/terms has to adhere to specified grammar rules and also form a standard adopted language. So in simple words, the Semantic Web is a web of information that is linked in ways to establish meaning that follows language constructs and a defined grammar.

In comparison to relational databases which depend on a schema, rows, columns, and keys for structure, a knowledge base comprises ontology, formal logic statements, and URIs. The statement is the foundation which consists of the elements which finally form a triple that contains a subject, predicate, and an object (for example: Adam isType Person), a URI provides a distinct name for the items and the ontology helps define the relationships, constraints, and concepts.

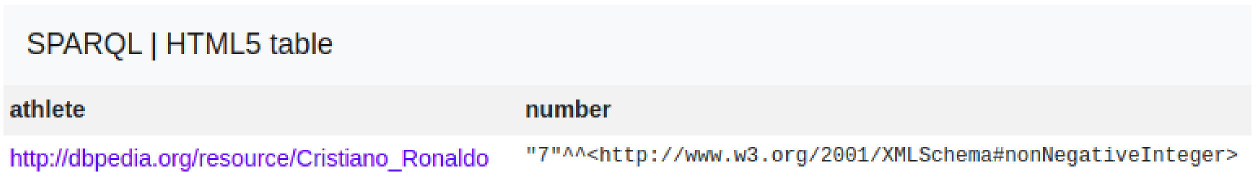
1.1 Querying the Information

Up to this point, at a higher level, we have discussed what Semantic Web is, what it contains and how information is stored. We would require a structured way in order to query the information from this large pool of data, and this discovery concerns a language that recognizes triples as the fundamental syntax, with which we can find required information that is stored in statements. For this purpose SPARQL is used, which is a standard set by W3C, to query the graph databases, also it has a vast community support. It is somewhat similar to SQL, as it uses keywords like SELECT, WHERE, DISTINCT, ORDER BY, etc.

With the growth of Semantic Web, grew its community which led to the creation of platforms like DBPedia, Wikidata, etc which are free and open knowledge bases. They contain the information which can be accessed using SPARQL queries. Here is an example of a SPARQL query and its answer.

```
SELECT *
WHERE
{
  ?athlete  rdfs:label  "Cristiano Ronaldo"@en ;
            dbo:number  ?number .
}
```

The above SPARQL query gives the jersey number of Chritinao Ronaldo. After hitting this query through the DBPedia's SPARQL Query Interface, we get the following response.



SPARQL | HTML5 table

athlete	number
http://dbpedia.org/resource/Cristiano_Ronaldo	"7"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>

Figure 1.1: Response from DBPedia's Query interface for the above query

From looking at the query as well as the response, one who does not have much experience working with SPARQL and Knowledge bases will find it difficult to understand them completely. So in order to simplify the whole process for any type of user to get the relevant information, a complete system is devised through this research.

1.2 Motivation

Graph databases or Knowledge graphs are growing in popularity and also complementing relational databases. Knowledge graphs are a strong concept for querying huge amounts of data, but these are typically enormous. They are often difficult to access or inconvenient to use for end-users because they require a expert understanding in query languages like SPARQL. And also demands a deep analysis of the structure/schema of the data models to be used.

This downside has led to the evolution of Question Answering systems that allow the users to provide their information needs in plain text and instead of a SPARQL query, a user can use natural language to search the database. The concept behind a Question Answering system over Knowledge Bases (KB), is to find the information requested by the user in an KB using natural language. This is generally addressed by translating a question in natural language to a SPARQL query that can be used to retrieve the desired information. While current systems simplify user access, there is still some scope for improvement in the performance and accuracy of these systems. In this dissertation, a system is proposed that implements the state-of-the approaches, evaluates them and also try improve the accuracy of the existing ways to create a QA system.

The proposed system will not only translate natural language questions into SPARQL queries but then fetch the results by hitting those queries on knowledge graphs in triples format and finally generate textual summaries from them as it is also not suitable for users to understand the answers in the RDF format as we saw in the example presented in 1.1.

An example of what such a system can do is shown below. Imagine if the user asks a question in natural language : *"Who was the creator of Mickey Mouse?"*. Not only will the user get a correct answer, but that will be in the human-readable format, which will be a summary in the english language like this *"Walt Disney was the creator of Mickey Mouse"*.

1.3 Question Answering Agent

Question Answering is an old research field of computer science. The most common type is finding the part of the text that contains the answer. Such systems are also called Information Retrieval systems, where a query is presented in natural language and using its keywords, matching documents are retrieved. We have already discussed above that question/answering (Q/A) agents allow users to ask questions in natural languages over a knowledge base. Nowadays, there are Knowledge Bases about publications, media, life-science, geography, etc. QA over Knowledge Bases has come out to make this freely available information available to the users easily.

To translate a natural language question to SPARQL, most of the work that exists takes a two-stage approach: (i) question understanding and (ii)query evaluation. Further generating a natural language summary from Knowledge Base triples is a separate task. When these two different tasks are combined as a pipeline, it becomes a Natural Language based Question Answering Agent over Knowledge Bases. In this research, we will focus only on the English language.

1.3.1 Question Translation

Multiple approaches exist for translating a question, including a rule-based system using a combinatorial approach, a graph-based approach to generate a semantic query graph, or even learning SPARQL templates using machine learning. The best possible system should allow the end users to benefit from the expressive robustness of Semantic Web standards like RDF and SPARQLs, while at the same time abstracting their complexity. Current systems in the question understanding stage translate a natural language question Q into SPARQLs, and in the query evaluation stage, evaluate all SPARQLs converted in the first stage.

1.3.2 Generating Textual Summary

Most people require textual formats to make sense or understand the Semantic Web data such as the RDF format triples. Though it is conveniently processed and accessible by machines, it is not convenient to be understood for users who are unknown of this technology. For humans, on the other hand, reading text or sentences in their native language is a much more accessible activity. In the context of the Semantic Web, Textual Summary Generation is related to the implementation of textual interfaces that would make the data/information stored in the form of knowledge base triples more approachable to the end users. A typical application would be this system's integration into a QA platform, whose user experience could be enhanced by the ability to automatically generate a textual description of an entity that is returned at a user's query.

1.4 Research Questions

The following research questions will be investigated to achieve the project's objective.

Question 1: Is it feasible to create a Question Answering agent over Knowledge Base

Graphs that takes a Natural Language question as input and returns a human readable natural language summary/answer for that question?

Question 2: Can we enhance the accuracy of the existing state-of-the art Q/A agent that converts a natural language question into SPARQL query?

1.5 Dataset

In this research, we focus on the "Question Answering over Linked Data" (QALD) benchmark, which is a series of evaluation campaigns and tasks for Question Answering agents(in this research which is regarded as query translation phase). It follows the principles of the Semantic Web: as it focuses on RDF Knowledge Bases that belong to different domains supposing that QA systems must be able to query any KB, it suggests a massive open-domain KB such as DBpedia, it contains questions with more than one operators and relations such as superlatives and comparatives. The data format used by QALD is JSON, as it is the most commonly used format and is easily understandable, and also allows programmers to focus on functionality and algorithm better. The training data from QALD challenge consists of 408 questions, where each question is represented as a JSON, which contains the natural language questions, its relevant SPARQL query and also its corresponding answer. For generating the textural summaries, we will be using Wikipedia summaries for the resources.

1.6 Dissertation Structure

The dissertation will be structured as follows: firstly, the background literature section, focusing on previous research discoveries that will aid in answering the research questions, including past research that was done in question answering systems over Knowledge Graphs, research done for converting SPARQL queries to a natural language representation and also how to generate textual summaries using Knowledge Base triples.

Then we move to the Methodology section, where the approach used to create such systems for this research is discussed in detail.

After that we have the Implementation section which talks about the datasets used, the complete architecture of the application and how the approaches mentioned in methodology was used and implemented. Finally we evaluate the systems implemented in this research and conclude it with Conclusions and Future Work section.

Chapter 2

Literature Review

This section will discuss the background literature related to this project. The main focus will be on how other research has tackled the Query Translation as well as the Textual Summary Generation process. Along with these two separate systems which are joined together as a pipeline, we will also discuss a system that converts a SPARQL query back into a natural language representation which will be placed in the middle of that pipeline. This step is introduced in order to enhance the accuracy of the chosen state-of-the-art query translation phase that is implemented for this research, and this phase will be termed as query refine step. Query refine phase is introduced here because while implementing the query translation phase, some limitations were noticed which led to such an additional step.

2.1 Query Translation

We review the Query Translation systems that participated in the QALD challenges or that were evaluated afterward using the same evaluation steps and used the same dataset as mentioned in the QALD challenges. The research that did the latter were fetched from Google Scholar that cited the publications of QALD challenges. In the introduction section we briefly described that query translation is majorly divided into two sections which is query understanding and query evaluation stage. In this section we will further divide these 2 sections into 4 tasks and review publications that cover them.

First is Question Analysis, in which the syntactic features of the presented question are analyzed which then helps determine which phrase corresponds to an instance(object or subject), deduce the correct segmentation of the question, also determine the class or property and the dependency between the separate phrases if any. The system proposed in Xu et al. (2014) decides in this step that which part of the natural language question will correspond to a class, instance or relation. Whereas other systems like Lopez et al. (2012) use this particular step to determine how the terms associate with each other and form a hypothesis about the relationships and the correspondence.

There are various techniques to achieve this question analysis step such as question segmentation using Part-of-speech tagging, recognizing the named entities and identifying dependencies using parsers. The task of segmenting a question corresponds to identifying the contiguous collection of tokens that refers to a particular resource. For example in the question "Who produced the film Forrest Gump?". Here the token "Forrest Gump" refers to a movie. One way to achieve this is by using a Named Entity Recognition (NER) tool. In He et al. (2014) it was observed that only 51.5 percent of entities could be recognized by the standard NER tool from NLP for the questions dataset in QALD challenge. Another approach that could be used to recognize entities is the n-gram strategy where n group of words are tried to map with the entities in the underlying Knowledge Base, this was used in He et al. (2014) and Shekarpour et al. (2016). This has a disadvantage that there will be multiple candidates and we will have to disambiguate them. To determine the named entities, systems like Daiber et al. (2013) and Yosef et al. (2011) use entity linking tools, which also finds the corresponding resource from the KB as they perform multiple steps of identifying tokens that refer to an entity, then look for the resources that are similar to them and then finally disambiguate them at once.

Part of speech tags are used mostly to determine which phrase of tokens relate to the objects, subjects or predicates. For example in the example "Who was the director of Top Gun", the noun 'director' can be referred to a property. The general idea is to determine Part-of-speech tag expressions, where they can easily be determined with the help of regex expressions. Some systems like Lopez et al. (2012), Freitas and Curry (2014) and Yahya et al. (2012) rely on regular expressions that are hand written. For example, Lopez et al. (2012) use a massive list of question templates, i.e to map a question to triples it considers the type of the question, its structure and the phrases. On the other hand, instead of using handmade rules, systems like Xu et al. (2014) uses ML algorithms to discover them. It uses a training set that consists of questions which are tagged with relations, entities, variables and classes. Finally a phrase tagger is built using this training corpus, where the features are word tokens in the questions, name entity tags, as well POS tags.

Next Step is Phrase Mapping, which finds a set of resources form the underlying Knowledge Base that corresponds to a phrase which contains one or more words. For example, the phrase EU could relate to the DBpedia resources University_of_Edinburgh or to European_Union or even Execution_unit. This can be done using the property rdfs:label which provides a human readable representation of the resource name. By using this convention from the RDF Schema, a resource can be related to a phrase where the resource's label contains or is equal to the phrase. Systems like Lopez et al. (2012) and Aggarwal and Buitelaar (2012) uses this strategy where the labels of the resources are indexed using Lucene which makes the search faster.

Another way to map a phrase with the label is to find semantic similarity between them. For this purpose a lexical database called WordNet can be used, where the strategy is to expand the phrase using the synonyms of phrase found in the lexical database. This is used in Lopez et al. (2012), Aggarwal and Buitelaar (2012) and Lopez et al. (2013), where the advantages are that more mappings can be resolved where the disadvantages are that mappings increases the number of possibly similar resources which then makes the disambiguation step computationally expensive.

Also by finding relations expressed in the natural language question, we can map that relation to a triple. For example in the question "Who is the director of Se7ev", first the object and subject are extracted which are 'Who'(variable) and 'Se7en'(resource), and they are related to each other using a relationship word 'director'. Hence we finally get the relation *director[Who, Se7en]*, which then can be mapped to resources in the KB. This approach is used in systems He et al. (2014) and Zou et al. (2014) and will be explained in depth in the further sections.

Next is the task of Disambiguation which helps find the exact resource from multiple options for a particular phrase. In order to rank them, feature such as semantic similarity is used, and in order to exclude some of them, type consistency checks are used. Systems such as Lopez et al. (2012), Aggarwal and Buitelaar (2012), Zou et al. (2014) uses graph based search to resolve the ambiguity.

Specifically in QA system Lopez et al. (2012), it considers the question analysis step to translate the question into a graph, where are each variable, class and entity is contained by a node, and relation is denoted by an edge. Now the ambiguity is carried over to the edges and vertices, where it is resolved by searching the sub-graph created from question in the Knowledge Base and then assigning a score to possible matches. This approach will be explained in depth in further sections as well. Apart from graph search, there are multiple other ways for disambiguation such as using Hidden Markov models, Integer Linear programming or even using user feedback.

Now we move to the task of Query construction where different methods of constructing a SPARQL query are described. But a problem called 'semantic gap' arises in this step. Let us consider the example question "Who is the mayor of New York", in this we will expect such a triple to exist in DBPedia.

```
:Eric_Adams :mayor_Of :New_York_City
```

But the actual triple that exists is:

```
:New York City :leaderName Eric Adams(en)
```

So this is a semantic gap which exists between the question and the Knowledge Base, and there are approaches to solve this problem such as forming SPARQL query based on fixed templates, constructing it using information deduced from the question analysis step, forming query using semantic parsing or even using machine learning.

Some systems use templates in order to generate SPARQL queries which are actually predefined queries with some placeholders that need to be filled. System such as Cabrio et al. (2012) restricts itself to only form the select query for only one single triple, whereas the system explained in Park et al. (2014) can only answer ASK queries, some SELECT queries with one or two triples as they only use a small set of templates. This technique has a disadvantage that not all type of questions can be treated with the help of templates. To create a form of a SPARQL query, most systems start with using the information deduced from the question analysis task. In Yahya et al. (2012), using regex over POS tags, triple phrases are selected during the question analysis task, then they are mapped to resources and disambiguated using Integer Linear Programming over the segments which finally returns the triples used to create the SELECT query. In the system presented in Zou et al. (2014), first of all the relations are deduced corresponding to the arguments, which is then used to create a vertices and edges of a sub-graph where this graph finally reflects the SPARQL query's structure.

2.2 Query Refine

After the query is constructed, the user has to rely on these translation systems in order to access the information in RDF format, irrespective of the correctness of the query, as these systems have not been able to explain the generated queries to the end user. This drawback has led to the creation of such systems that convert SPARQL queries back to a natural language representation.

For system presented in Zou et al. (2014), consider the sample question "What is the data of birth of Jay Cutler", where it returns more than 15 possible interpretations, including for thei birth date of weightlifter and four time Mr.Olympia Jay Cutler as well as the age of pro American football player Jay Cutler, where each interpretation is a SPARQL query. A lay user would not be able to pin point for which person they want the answer for by just looking at the queries, they would be more interested in a verbalised form of that query and then pick their choice. In this research we will not take a user's feedback, but use this system and fit it into our QA system's pipeline that will automatically feed the desired query to the query interface to get the answer in triples format.

We will review a few researches done for this type of system. First is SPARQL2NL Ngonga Ngomo et al. (2013), which consists of a 4 step approach, a pre-processing step in which the query is normalized and using its variables, extracts the type information, then a processing step during which just a generic NL representation of the query is constructed, the a post-processing step where the legibility of the NL representation is improved by applying replacement and reduction rules and then finally a realization step which generates the natural language verbalization of the query. A demo of this approach is show below:

```
Question: What is the capital of Yemen?
NL representation : The query retrieves Yemen's capital. The query
                    returns 1000 result.
```

Another approach is SPARQLtoUser Diefenbach et al. (2017), which is not restricted to a single Knowledge Base and is multilingual too. To generate the representation, the actual order of the triples is kept as it is in the query and the URIs are substituted with their labels. Then, if a predicate is also a URI, it is simply replaced with its label otherwise if it is a variable, then it is expanded using all the properties that could potentially fit in that position.

2.3 Textual Summary Generation

Natural Language Generation(NLG) systems usually work in 3 phases: first is content selection in which the data that will be conveyed in the text form, is selected and structured. The output of this phase is then presented to the next phase which is micro-planning, which decides how this information has to be linguistically displayed in the created text. Then there is the surface realization phase, which takes in the linguistic requirements set by the micro-planner and then finally generates the actual text satisfying those requirements.

A system that relies on these three phases is explained in Lapalme (2020), where the first step in textual summary generation is identifying the information to be conveyed in the text, which is the set of triples. As the predicate of a triple specifies a relation between its object and subject, in this research's case, it is mapped to a verb linking the object and the subject of the sentence.

There are various text generation systems that completely rely on the predefined rules where the phases mentioned above are performed independently as they are associated with the language and the domain of the application as applied in systems Green (2006), Turner et al. (2009) and Reiter et al. (2005). These kind of systems do not adapt well to a large set of domains.

Most of the work done for the textual generation systems addresses this as a sequence-to-sequence problem. And this sequence-to-sequence task is mostly solved using Neural Networks and Deep Learning. Other than that, neural networks are also applied in the tasks related to NLP that ranges from automatic response, machine translation or even textual descriptions generation from visual data/images. In systems like Zhu et al. (2019) and Vougiouklis et al. (2018), an encoder-decoder architecture is used to generate sequences of the required textual summary from the triple. Whereas Gao et al. (2021) joins a graph-based encoder as well to the seq2seq, in order to model the RDF triples and finally decode and textual sequence.

Chapter 3

Methodology

3.1 Query understanding and evaluation

For this research, we will consider the state-of-art approach to implement the Query Translation Phase explained in gAnswer Zou et al. (2014). According to QALD challenge 9 Ngomo (2018), gAnswer Zou et al. (2014) performed best amongst all the submissions. It uses a graph-based methodology in order to create a semantic query graph, which brings down the problem of transforming the natural language question to SPARQL to a problem of matching sub-graph to a bigger KB.

Due to the unstructured nature and the ambiguity in the natural language question, the hardness level for creating a Query Translation system increases. The 2 major challenges are *Phrase Linking* and *Composition*. A natural language phrase may have multiple meanings or it could be related to various resources in the Knowledge Base. For example in the question "What is the cast of the movie directed by Paul Anderson?", the phrase 'Paul Anderson' is eligible to be mapped to more than one resources such as Paul_W_S_Anderson, Paul_S_Anderson, or even Paul_Anderson(actor). Also the relation 'directed' can be mapped to multiple possible properties like (writer), (director) or even (directed_by). After identifying the resources or mapping them to phrases, the task is to assemble the query or the query graph by putting these phrase together. In the current example, we can use the director to link the object and subject and get the desired relationship as (?movie, director, Paul_W_S_Anderson). However it is sometimes difficult to find such a relation or there may even exits multiple possible sub graphs for the query. So we will have to deal with this ambiguity as well.

The approach used in this research combines the query disambiguation and query evaluation stage together. Specifically, during the time when matches are being found for a query, ambiguity for the natural language question is resolved, as this saves the cost of disambiguation and refuses it from being an extra and separate step. So at higher level, there are 2 stages in this approach which are *query understanding* and *query evaluation.* By resolving the ambiguity at the evaluation phase, this gives the liberty to create a query graph from the user's question and allow ambiguity during the phrase mapping phase which can be resolved later in an efficient way.

3.1.1 Relation-First Framework

We can apply a relation first framework here, in which, first the dependency tree structure is created and based on that the semantic relations are extracted from the question, which are then used to build the semantic query graph. Here a triple *(sub, rel, obj)* is considered as a semantic relation, where sub and obj are the node phrases and rel is the relation phrase. Considering the above triple example again, for the triple (?movie, director, Paul_W_S_Anderson), this framework assumes that it will either find the graph with exact match and remove the ambiguity at the question understanding step itself or it wont find any graph at all and also does not address query graph's structure's ambiguity. This framework often fails when there is any uncertain or an implicit relation in the natural language question.

3.1.2 Node-First Framework

The framework that takes another perspective is the node first framework. This one does not fail to extract the implicit relations as it starts by identifying the nodes from the NL question and then connecting them to generate a query graph. Also this framework allows for ambiguity in the query graph structure, as opposed to the relation first framework. In the question understanding phase, this framework builds a super-graph of the required query graph, where there exists uncertain edges. Finally the ambiguity is resolved when the super-graph is matched over the underlying RDF Knowledge Graph and finding correct matches during the query evaluation phase.

With the help of an example, we can understand the idea behind node first framework. Consider the question *"What is the release date of the movie directed by Christopher Nolan and starred by an Australian actor?"*. This would have 2 additional triples, which are (?movie,starring,?actor) and (?actor, country, Australia) in addition to a triple of (?movie, director, Christopher_Nolan) which can be extracted even using relation first approach. But the triple that contains the relation 'country' to the 'actor' would not have been extracted by relation first framework as it is not explicit in the question's intention.

Using this framework, an edge is created between the vertex v4 (actor) and the vertex v5 (Australian) as seen in fig 3.1 and there is no label on that edge. For the identified relation (starred by) it is not straight forward to determine its corresponding nodes as it has 3 candidates: actor, movie and Christoper Nolan. In this graph, an edge is introduced between movie and actor and also between Christopher Nolan and actor. During the query evaluation phase, the match between the super graph is found with RDF Knowledge Graph and a final match is made. In that final match, the edge between v3 and v4 is vanished (relation between Christopher Nolan and actor) as it does not exist in the underlying RDF Knowledge Graph.

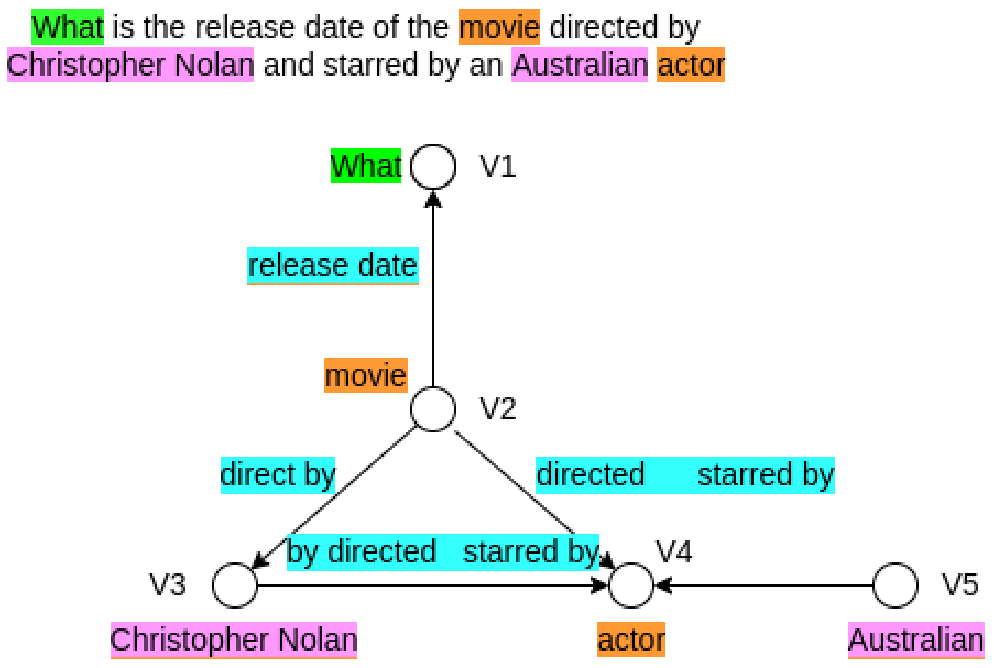


Figure 3.1: Creating a Super Semantic Graph for the given question

Through the above example we understand that node first framework tries to fix the semantic query graph when the matches are identified, whereas the answer to the question have already been found. In simple words, at the time matches are being found the structure ambiguity of the query graph is resolved.

Now lets understand how the phases of Query Understanding and Query Evaluation work under this node-first framework.

Query Understanding

Provided a question sentence in natural language, entity extraction algorithms are applied in order to extract all nodes from it which are referred to as classes or entities. Also the variable nodes are extracted which are the wh words (for example Who, Which and What). Then if there is a semantic relation between any two nodes then an edge is introduced between them, which is then used to build a super semantic query graph for that question. The introduction of an edge between two nodes is a naive solution and has exponential time complexity as it introduces noise and may lead to more unnecessary edges. A better approach would be to use the dependency parse tree with the following assumption.

***Assumption**: A semantic relation is built between two nodes v1 and v2 iff, there is no other node that exists between the path of v1 and v2 in the dependency tree of the textual question.*

Let use recall the example, *"What is the release date of the movie directed by Christopher Nolan and starred by an Australian actor?"*. In this, we first extract the nodes which are: 'what', 'movie', 'Christopher Nolan', 'actor' and 'Australian'. Fig 3.2 shows the dependency parse for the above sample question. Corresponding to the above assumption, we introduce an edge between node v1 and v2 as there is no other node that lies between the path between v1 and v2. For example in the super graph, we can see that the edge label is '(be)release date' between nodes 'What' and 'movie'. For nodes 'Christopher Nolan' and 'actor' there is no other node between them in the parse tree. We can also observe that there is an implicit relation between 'actor' and 'movie' which is connected by the edge 'directed starred by' according to the assumption above. And finally we obtain this super semantic query graph for the NL question.

Query Evaluation

For each edge and node, candidates are found in the super graph by using the entity mention dictionary (explained in section 4.1) which is created by using the surface string(rdfs labels) corresponding to that entity. Each edge label is also mapped to possible candidates which are predicates in the relation mention dictionary. The edge between 'Christopher Nolan' and 'actor' is not found in the Knowledge Graph, hence it is not matched and is not a triple in the semantic query graph.

3.2 SPARQL to Natural Language

In this research, this step is not a separate or independent step, but additions to the phrase mapping as well as query disambiguation phase for the query translation system.

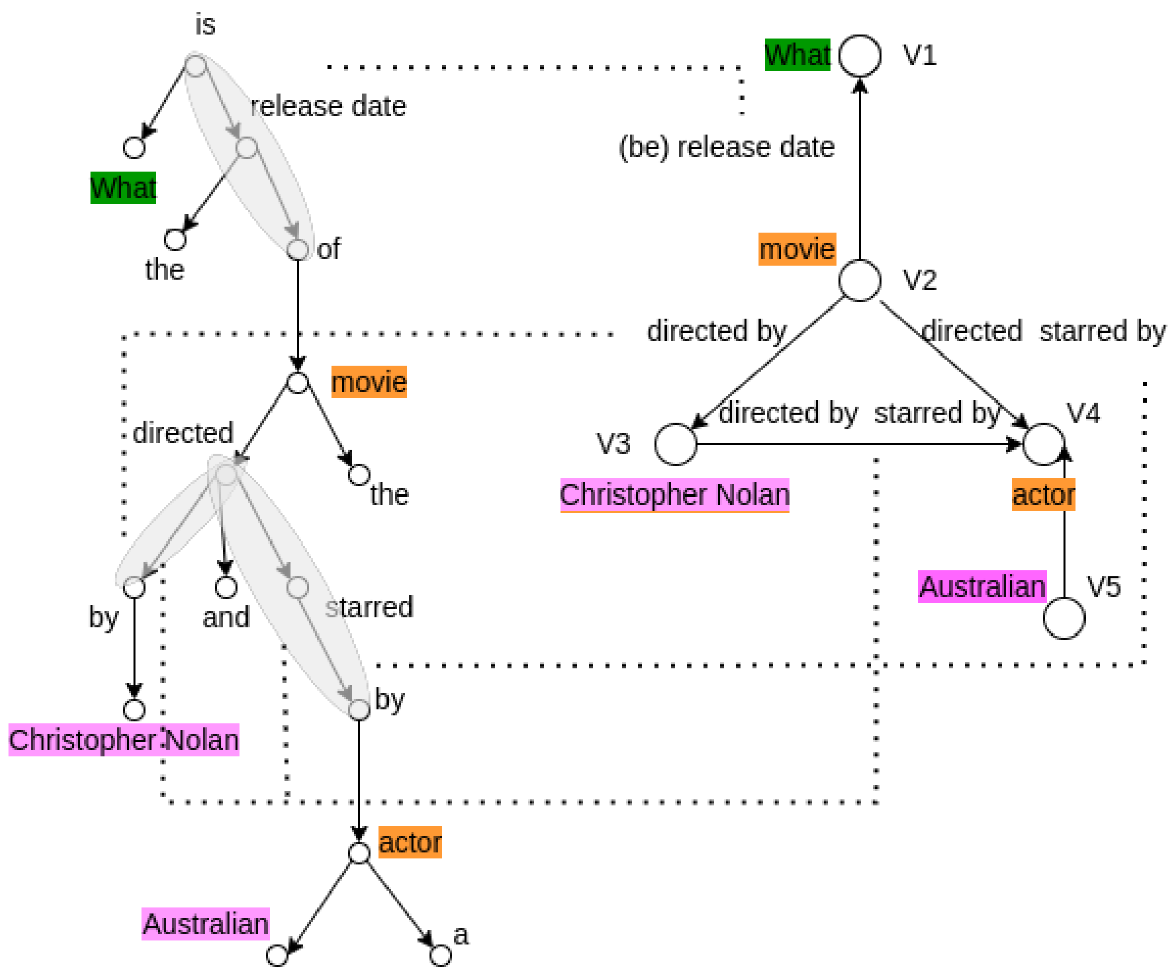


Figure 3.2: Dependency Parse Tree and a Super Semantic Graph for the given question

During the step of extracting nodes or entities from the NL question, there are multiple instances where there is no match between the phrase and the resource in the Knowledge Base. This leads to a wrong answer or no answer. For example let us consider the question *"Who is the mayor of New York?"*, there exists no property related to 'mayor' on the DBPedia page of 'New You City', and hence we are not able to generate a query for this question. But there is a property of 'leaderName' which tells the actual mayor's name and which is also the required answer to the question.

So this led to an idea of using synonyms if there is no match for a question. We can try synonyms of the word 'mayor' in this example, and one of the synonyms is 'leader' and hence the question will turn into *"Who is the leader of New York?"*. Then using this newly suggested question by the use of synonyms, we try the phrase mapping phase again. With trying multiple synonyms for each possible node, it may lead to multiple candidates and it will require disambiguation of a different type as mentioned from the above sections. So the idea is to construct SPARQL queries for all the new questions, then convert each query back to a verbalised representation and check its semantic similarity with the original question that was presented by the user. Whichever has the highest similarity score, that one SPARQL query corresponding to it will be used to answer the question.

Now lets see how a SPARQL query is converted to a verbalized form that a lay user can understand easily. The original pattern of the triples in the SPARQL query is kept as it is to generate the representation. The URI's in the query are substituted with the label in the corresponding language (which is English here). A concrete example is shown below:

```
PREFIX onto:<http://dbpedia.org/ontology/>
PREFIX res:<http://dbpedia.org/resource/>
SELECT DISTINCT  ?uri
  WHERE
     { ?uri  onto:director  res:Clint_Eastwood ;
             onto:starring  res:Clint_Eastwood
     }
```

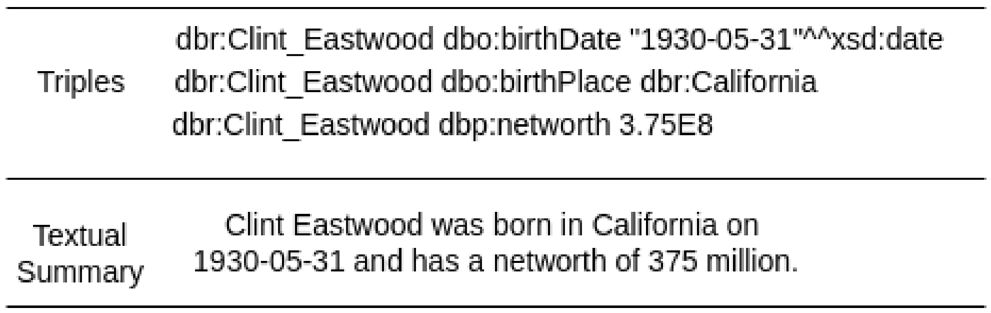
is converted to :

film director / Clint Eastwood / starring / Clint Eastwood

The substitution of the predicates depends on whether it is a variable or a URI. If it is a URI then just replace it with the label. If it is a variable, then it is expanded using all the properties that could possibly be placed in that position.

3.3 Text Generation from SPARQL query

An ideal example of a natural language generator system is shown in the figure below. This system takes a set of triples about Clint Eastwood as an input, whose either object or subject is related to the entity of Clint Eastwood in DBPedia, and gives a textual summary as an output that is generated by sequence of words.



Triples	dbr:Clint_Eastwood dbo:birthDate "1930-05-31"^^xsd:date dbr:Clint_Eastwood dbo:birthPlace dbr:California dbr:Clint_Eastwood dbp:networth 3.75E8
Textual Summary	Clint Eastwood was born in California on 1930-05-31 and has a networth of 375 million.

Figure 3.3: Example of what NLG task is required for.

Model

Given a set of N triples, T : [t_1, t_2,...t_n], we have to create a model that can learn and is able to create a sequence of Y tokens, G = [g_1, g_2,...g_y]. Here G is regarded as the natural language representation of the triple set in the input. The requirement is to build a model that is capable of calculating the probability for generating G from T.

The model comprises of a feed forward architecture, which is a neural network that consists of the input nodes, hidden nodes and output nodes. The input data flow is only in forward direction and never backwards in order to get processed as there are no loops in this type of network. The feed-forward architecture encodes all the triples one by one from the input set using an encoder, into a vector form which has a fixed dimensionality. This encoder is then accompanied by a Recurrent Neural Network(RNN) based decoder which helps generate the actual textual summary sequence by sequence or token by token. The model's architecture is shown in Fig 3.4.

In this architecture, we can see that the encoder takes triple set as input and then converts them one by one into a vector representation, V_{f1} and V_{f2}. After this, all the vectors are concatenated(V_f=[V_{f1}:V_{f2}:..V_{fn}]) and presented as input to the decoder in

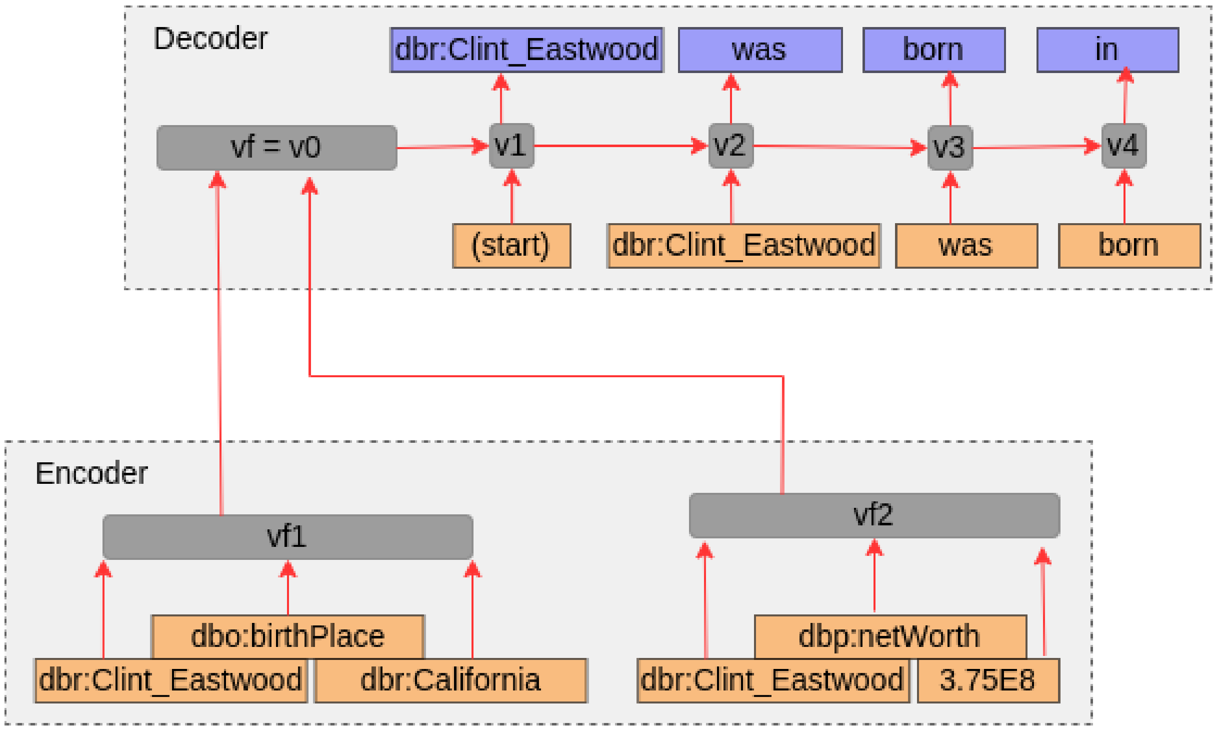


Figure 3.4: Encoder Decoder architecture

order to start the decoding process of summary generation sequence by sequence. There are predefined start and end tokens for every generated text, start of the sequence is displayed by the (start) token and end of the sequence is show by (end) token. At every time-step *a*, decoder takes the current word as an input along with the state in the time-step *a-1* and using both of these, a prediction is made based on probability what will be the next token in the sequence. For example, decoder is at v2, the current input is dbr:Clint_Eastwood and also takes the previous state from h1, combined them both and predicts the word was as the next token with the highest probability.

Encoder

In order to compute the vector representation V_{fi} of a triple t_i, the encoder is forward propagated. In forward propagation, the input data moves in the forward direction, and it is processed according to the activation function in the hidden layer and then passes the processed data to the successive layer. Here the processing is done in 2 steps: *preactivation* - which is also the weighted sum of the inputs and *activation* -in which the calculated sum passes through a mathematical function that adds non-linearity to it.

We have a trainable weight matrix(W_x), that is learned from providing the training dataset to the neural network. Also we do one-hot encoding on the triple's subject, predicate and object and get their one-hot encoded form as s_i, p_i and o_i respectively. We then calculate the weighted sum (S_{fi}) for that triple using the following formula:

$$S_{fi} = W_x.s_i; W_x.p_i; W_x.o_i$$

After getting the weighted sum using the *preactivation* step, we apply the activation function to S_{fi} and finally get the vector representation(V_{fi}) as the output. The activate function used here is ReLU, which is a ramp function and adds non-linearity to the input. We finally get vector representation using the following formula:

$$V_{fi} = ReLU(W_h.S_{fi})$$

where W_h is a different weighted matrix learned for the weighted sum.

Decoder

The vector representations for all the triples are concatenated into a single vector since they are not sequentially correlated. This concatenation based formulation helps capture the information over all the input triples that are given as input. After we have the single concatenated vector, we calculate its weighted sum using a weighted matrix and we finally get the vector representation (V_F) that will be passed to the decoder.

Now the decoder makes a prediction at every time-step a, for the next token that will be appended to the final generated text by taking into consideration the already generated tokens and also context knowledge from triples and their vector form. We use a multi-gated RNN for this purpose as they have the ability to process longer sequences of information as compared to the simple one. In this research, we have used a Gated Recurrent Unit adopted from Chung et al. (2014), which is a type of multi-gated RNN that process this information.

For each token in the summary, the conditional probability is calculated using the softmax activation function over all the elements present in the dictionary of textual summaries. Softmax function is generally kept as the last layer of the network, which normalizes the output to a probability distribution. Such approach does provide an optimal summary, but the fact that it requires a large target vocabulary makes this task unmanageable.

Another approach could be to approximate the best summary by picking up the token with the highest probability. While these type of greedy decoders prove to be fast, but the quality of the output is considered to be low.

A compromise between these 2 could be to use a beam search decoder, which provides M most probable summaries for the given set of input triples. And at a given point of time, the decoder only maintains small number of M incomplete summaries which are extended at each time-step with a new token when it is appended.

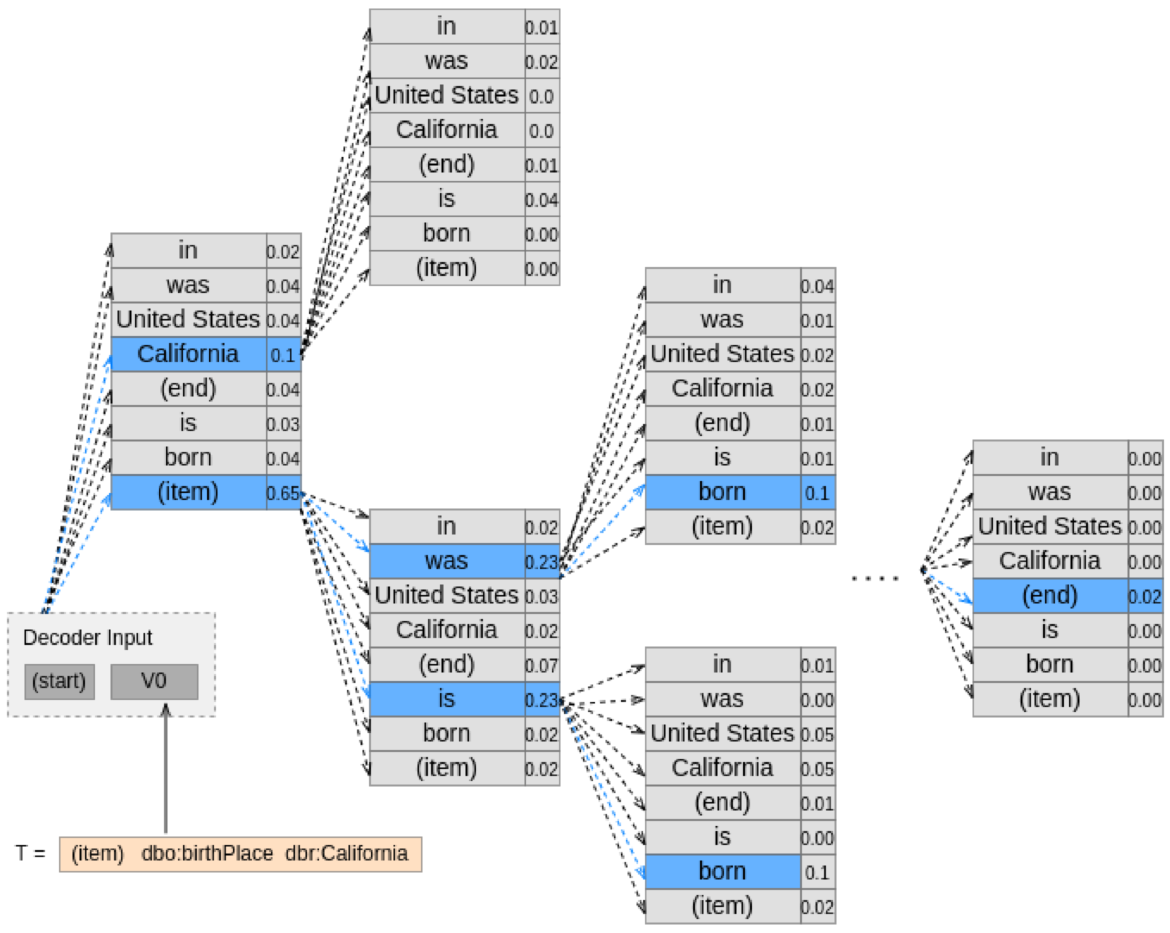


Figure 3.5: Beam Decoder Example

An example of the beam search decoder can be seen below in Fig 3.5, which contains beam M of size 2 and the target vocabulary of size 8. Decoder starts with taking the concatenated vector representation of the all the input triples set and also adds a token (start) to the output summary. When the summary is extended by appending a particular token to it, that token gets a probability which is shown besides the words in the vocabulary. In time-step 1, the decoder keeps the two most probable words from the vocabulary with which a summary could be started. At the 2nd time-stamp, (item) and *California* are given as input to the decoder which results in 16 partial hypotheses. Due to the inadequacy of occurrences of the majority of the entity of interest in the datasets, it would not be possible to identify a distinct vector representation for those entities of each Wikipedia summary, so they are replaced with a special (item) token. Similar to Wikipedia summaries for an entity, the occurrence of the main entity of the corresponding summary as either object or subject of a triple is represented with the special (item) token.

At a given time, we only move forward with 2 summaries, and since both of them start with the (item) token, we discard the token *California* as the first token after (start). At the next time-step, *was* and *is* are separately provided to the decoder as they have the highest probabilities. This process continues till we predict the (end) token. At the end, complete hypotheses is generated as following:

```
1. ‘ (start) (item) was born in California (end)’ with
   a probability of 0.04
2. ‘ (start) (item) is born in California, United States
   (end)’ with a probability of 0.02.
```

After we have received the summaries from the decoder, we just remove the unnecessary tokens or replace them with the entity labels and finally pick with the highest probability.

Chapter 4

Design and Implementation

4.0.1 Development Environment

1. **Operating System** : Ubuntu 20.04.4 LTS
2. **IDEs** : Eclipse, Jupyter Notebook
3. **Programming Languages** : Java, Python
4. **Frameworks, Databases** : Spring Framework, Lucene, GraphDB Triplestore

4.1 Datasets

There are more than one systems that are combined to create the desired Question Answering system for this research. Each type of system is different from the other and requires a dataset of their own. The Query Translation system depends on the DBPedia's Knowledge Base as well as its data in the triples format. Whereas the Textual Generation system requires training data as it learns a neural network model for this purpose. The training data comprises of RDF triples and their corresponding entities' Wikipedia summaries.

Entity Mention Dictionary

The surface form of a string that refers to entities is an *entity mention*. For example, the phrase "Michael Jordan" could refer to the basketball player (Michael_Jordan) or the actor (Michael_B._Jordan). A table such as 4.1, is required in order to lookup for the entities which have their corresponding confidence scores as well using the string phrases provided in the natural language question.

Entity Mention	**Corresponding Entity**	**Confidence Score**
"Michael Jordan"	(Michael_Jordan)	0.85
"Michael Jordan"	(Michael_B._Jordan)	0.7
"America"	(United_States)	1.0
"U.S.A"	United_States)	1.0

Table 4.1: Entity Mention Dictionary with confidence scores

There is a lot of research that exists in this field such as in Spitkovsky and Chang (2012) and Chisholm and Hachey (2015). Crawling Web pages and aggregating the anchor links which points to the Wikipedia's entity pages is a popular way to create such a dictionary. The score here can be calculated by calculating how frequently the mentioned text on web page links to an anchor link. In this research we would be adopting an already built entity mention dictionary called CrossWikis dictionary from Spitkovsky and Chang (2012) which contains more than 180 million entity-strings mappings which were computed using a Google Web crawler.

Relation Mention Dictionary

The surface form of a string that takes place between two entities is a *relation mention*, such "director of" and "created by". In order to map the relations mentioned in the question to some predicate, we would require a relation mention dictionary as shown in table 4.2.

Relation Mention	**Corresponding Predicate**	**Confidence Score**
"place of birth"	(birthPlace)	1.0
"created"	(creator)	0.75
"starred by"	(starring)	0.9

Table 4.2: Relation Mention Dictionary with confidence scores

In this research, we do not discuss how to extract relations from a string phrase, as it is a separate research problem and we assume that relation mention sets are already given. There is a lot of NLP related literature that studies and solves this problem of relation extraction such as in Fader et al. (2011) where in order to extract them it uses the n-grams approach.

QALD Dataset

In order to evaluate the Query Translation phase and check whether the query formulated is correct or even the answer received by hitting the query on the Knowledge Base is correct or not. This dataset contains 408 questions in total, each question contains the natural language question in multiple languages, a corresponding SPARQL query, and also the answers that we get after hitting that query on a Knowledge Base.

Knowledge Base

The Knowledge base is used in the research to create a semantic query graph using the provided NL question. The underlying Knowledge Base used here is DBpedia version 2016-10 dataset. Altogether the DBpedia 2016-10 release contains 13 billion pieces of information (RDF triples) out of which 1.7 billion were extracted from the English edition of Wikipedia, 6.6 billion were extracted from other language editions and 4.8 billion from Wikipedia Commons and Wikidata. So ultimately, we would require the triples that are only in English language as our system only supports English. Hence this knowledge base would contain 6.5 billion triples in total.

Apart from the triples data, we are also using the Mapping-based Object dataset which contains the DBpedia triples and their corresponding Wikipedia summaries, Mapping-based Literals dataset that retains articles only for which an infobox exists and the Instance Types dataset that is used to map the entities that occur rarely in the aligned datasets to special tokens. These will be used in the textual summary generation task as they make up the training data for the neural network model.

4.2 Design

Following is the design of the architecture for the application that we are trying to build through this research in Fig 4.1. The flow is as follows:

1. First the user provides with a question in natural language (English). In this example, the question is "Who is the mayor of New York?".

2. Next, the NL question is passed as an input to the query construction phase, where are query is constructed and evaluated over a Knowledge Base.

3. Inside the query construction phase, first the entities are extracted from the phrases, then a super semantic query graph is created.

4. The super semantic graph is passed to next stage for query disambiguation, where the best query is selected and hit on the Knowledge Base to get the answer.

5. There are 3 possibilities for the query construction phase, (i) there was only one query was created, in this case we pass the answer to the Natural Language Generator task to convert the answer triples in textual summary and then finally to the user, (ii) there was no query created, in this case we apply the synonyms approach due to which we will get multiple NL question substitutes and pass all of them to the query construction phase, (iii) if there are multiple queries, this case arises because of using the synonyms approach, to resolve this problem we convert SPARQL queries to a NL representation and check their similarity scores with the original question provided by user, and pick the one with highest score.

4.3 Implementation

The complete application is built behind an Application Programming Interface (API). The user will access this api, providing a natural language question in English as a query parameter, and the rest is done using the pipeline mentioned in the design section above. Finally the user gets a a natural language summary as an output from the application as a response. We have created such a system in order to abstract the system's complexity from the user, where they can simply access the system to get answers for their questions.

Recognizing Nodes

Recognizing the nodes from the input question is the very first step. All the classes, entities or variables are extracted as nodes. For this stage, the Entity Mention Dictionary is used to find the classes and entities. In the example: *"What is the release date of the movie directed by Christopher Nolan and starred by an Australian actor?"*, the extracted nodes are "what", "movie", "Christopher Nolan", "Australian", "actor". This is done by performing tokenization of the sentence and removing the stop-words as well, which is achieved using the NLTK library.

For each token/phrase, we look up the entity dictionary, and based on the confidence probability, pick up the most suitable entities. This lookup is made fast by using Lucene library over the entity dictionary, where all the entity mentions are indexed, which makes the search faster.

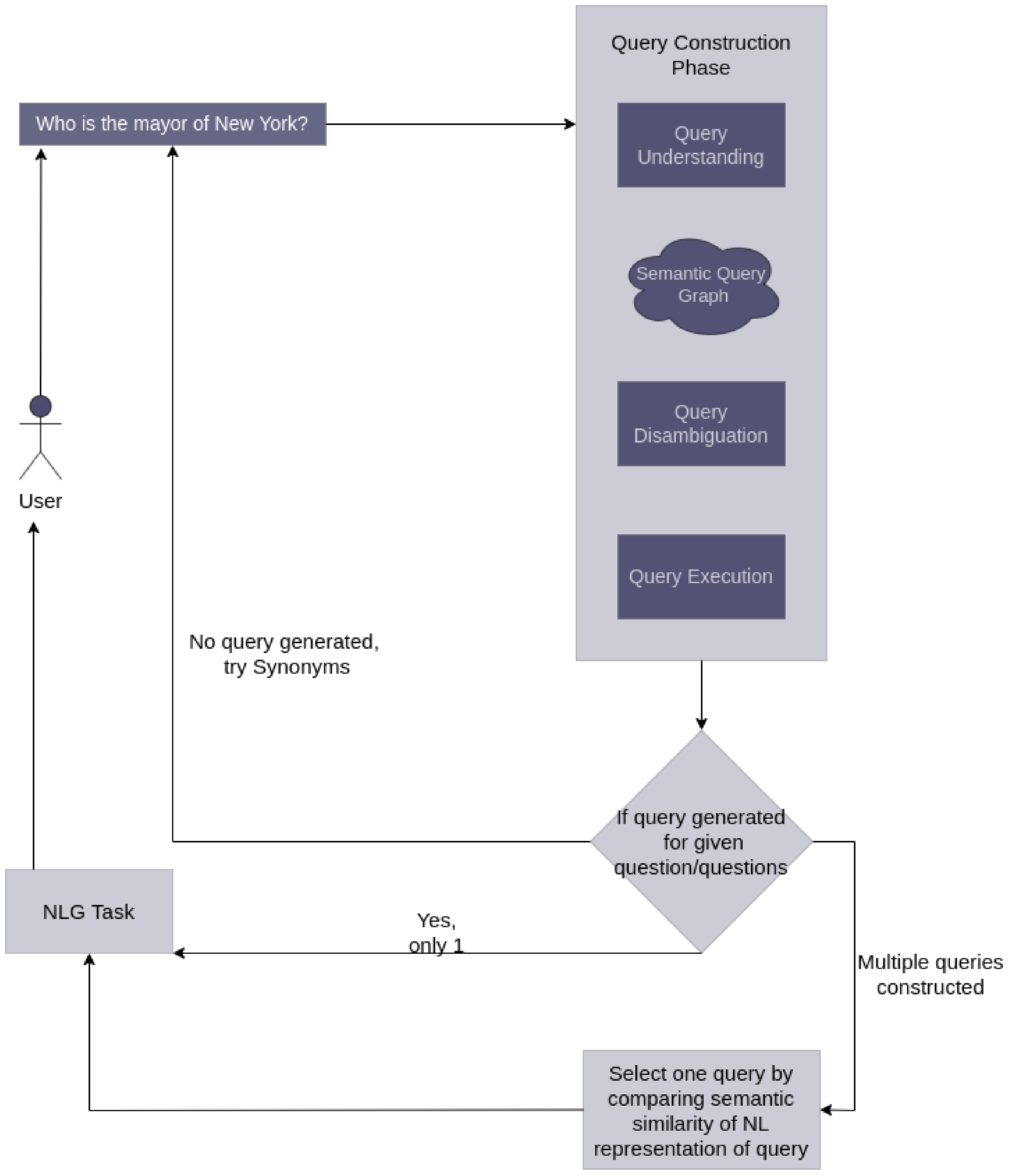
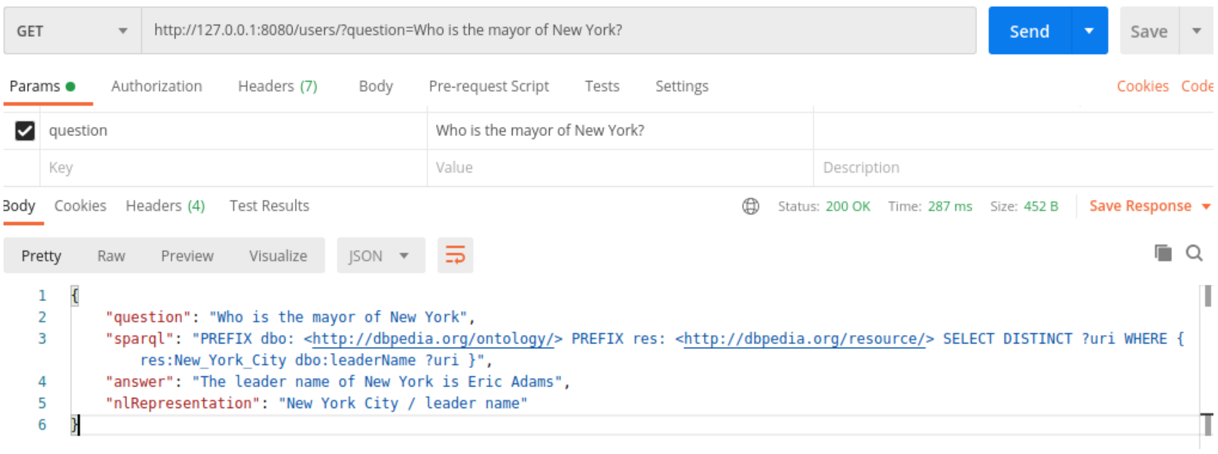

Query Construction Phase
Who is the mayor of New York?
Query Understanding
Semantic Query Graph
Query Disambiguation
Query Execution
User
No query generated, try Synonyms
If query generated for given question/questions
NLG Task
Yes, only 1
Multiple queries constructed
Select one query by comparing semantic similarity of NL representation of query

Figure 4.1: Question Answering System Architecture



```
http://127.0.0.1:8080/users/?question=Who is the mayor of New York?
```

	Key	Value	Description
✓	question	Who is the mayor of New York?	

Status: 200 OK   Time: 287 ms   Size: 452 B

```
{
    "question": "Who is the mayor of New York",
    "sparql": "PREFIX dbo: <http://dbpedia.org/ontology/> PREFIX res: <http://dbpedia.org/resource/> SELECT DISTINCT ?uri WHERE {
        res:New_York_City dbo:leaderName ?uri }",
    "answer": "The leader name of New York is Eric Adams",
    "nlRepresentation": "New York City / leader name"
}
```

Figure 4.2: Screenshot of the application API and its response

Structure Construction

After all the entities have been recognized, building the super semantic query graph is the next step. In order to do this, first the dependency tree for the question is created, which is achieved using part-of-speech tagging with the help of NLTK library. The approach used to build the super graph from dependency tree has already been discussed.

In order to implement it, Depth First search algorithm is used. It takes one node which is represented by one vertex, and checks the path to another node represented by another vertex. If there is no other node that exists between those 2, then we can establish a relation between them by adding an edge.

Phrase Mapping

Not just entity mapping, but same goes for the relation mapping as well where it helps find the predicates. This is done with the help of the relation mention dictionary which is also indexed using Lucene library.

For any implicit relations is the NL question, there exists and unlabeled edge in the step for creating the super semantic query graph. So in phrase mapping, we also focus on mapping the predicates to unlabelled edges. For an unlabeled edge, an implicit relation exists between the two nodes of that particular edge. From the example *"What is the release date of the movie directed by Christopher Nolan and starred by an Australian actor?"*, we can observe that "Australian" and "actor" nodes have an implicit relation. Based on underlying knowledge we infer the implicit relation, and then try to map it to a predicate.

1. The distance between the nodes which have an implicit relation, is short enough in

the RDF Knowledge Graph.

2. Atleast one of the nodes in this relation is an entity, if they both are wh-words then it is not possible to extract the relation.

Possible candidate predicates are found using the Knowledge Graph. If both the nodes are entities, then we identify both the nodes in the Knowledge Graph, and then find the predicate that lies between them. If one of them is a node and other is variable, then we first locate that one node in the Knowledge Graph and based on frequency we pick the candidate predicates that are adjacent.

Query Execution

Provided with the semantic query graph, we now have to find the best matches of vertices and edges with the RDF Knowledge Graph. The requirement is that all nodes should match, but some edges can be left if no predicate exists in the Knowledge Graph. In order to find match between the 2 graphs, we use a bottom up approach. Here is the algorithm for that approach from Zou et al. (2014), which takes the RDF Knowledge Graph and the super semantic query graph as inputs and gives the matched between both these graphs as output.

```
1:      Q ← start node n
2:      queue.push(n)
3:      while x = queue.pop() do
5:          for each vertex  E(QU )  v-x if does not belong to E(Q) do
6:              Q = Q union vertex
7:              if GraphExplore(G, Q) find matches then
8:                  if Q is a spanning subgraph of QU then
9:                      Insert matches of Q into answer set RS.
10:                     Only keep the matches in RS with the top-k match
11:             else
12:                 Q = Backtrack(Q, v-x)
13:             if v-x  Q then
14:                 que.push(v)
```

`In this algorithm, QU represents the super semantic query graph,`
`G is the RDF Knowledge Graph, v-x represented the edge between v and x.`

We keep on building a partial structure Q, whenever we find a match. To start, we take the vertex with least number of candidates which is the s here. In lines 7-10, if we keep finding matches, then keep appending nodes to the partial structure, otherwise backtrack and go check for other nodes.

Once we have the final graph, that contains the disambiguated triples, we construct the final query using a the query templates, where the placeholders in templates are replaced by the nodes of the final graph.

Trying Synonyms

There are instances where for a given NL question, there is no matching query generated. This could be mainly due to no mapping found for the phrases in the entity mention dictionary or the relation mention dictionary. This problem can be addressed by substituting the tokens in the question sentence by their synonyms.

Consider the example question "Who is the mayor of New York?", where no query was generated by this system because there is no property for mayor on the DBpedia page for (New_York). First the sentence is tokenized and stop-words are removed, then the tokens left are : ["mayor", "new", "york"]. For these tokens list, we take 6 synonyms each and try to replace them with the actual word. Initially we tried with the wordnet library, but it was noticed that the number of synonyms are quite low and relevant synonyms were missing. So we tried for the synonyms from Dictionary.com by web scraping it.

Following is the list of questions that we get when we replace the original word with its synonyms. We have 3 tokens to be replaced, and for each token we are tying 6 token, so we will get 18 new question sentences.

```
['Who is the minister of new york city ?',
 'Who is the superintendent of new york city ?',
 'Who is the dean of new york city ?',
 'Who is the chief of new york city ?',
 'Who is the judge of new york city ?',
 'Who is the leader of new york city ?',
 'Who is the mayor of brand-new york city ?',
 'Who is the mayor of unique york city ?',
 'Who is the mayor of current york city ?',
 'Who is the mayor of newly york city ?',
 'Who is the mayor of different york city ?',
 'Who is the mayor of modern york city ?',
 'Who is the mayor of new york metropolis ?',
 'Who is the mayor of new york boom town ?',
 'Who is the mayor of new york downtown ?',
 'Who is the mayor of new york capital ?',
 'Who is the mayor of new york municipality ?',
 'Who is the mayor of new york borough ?']
```

Now, same steps of query construction are carried out for all the questions. It is possible that we may get one query each per question or no query for some of them. Lets say we got a total of 11 queries from these 18 questions which are completely valid and can get some answer but not sure whether which one will give the desired answer, so now we have to disambiguate them.

SPARQL to Text

When we have multiple queries, in order to disambiguate them, we covert them to a natural language representation. For example for the new question

(i) *"Who is the leader of new york city ?"*, we get the NL form as ***"New York City / leader name"*** and for another question

(ii) *"Who is the superintendent of new york city ?"* we get the representation as ***"New York City Hall / superintendent / location"***.

In order to generate this NL representation, we are using the rdfs:label property of the resources.

For these 2, when we calculate the text similarity (which the cosine similarity between the vector form of the two texts), with the original question by the user, we observe that (i) has highest similarity score amongst all the other questions. Hence we choose the NL question as *"Who is the leader of new york city ?"*, and it also gives the correct and required answer.

Converting triples to text

We have a dataset of Mapping-based Objects from DBpedia which contains the DBpedia triples and their corresponding Wikipedia summaries. But the challenge here is how to align the triples with the lines in the summaries and also identify in the text how the entities of the KB are mentioned. For example in the Wikipedia sentence "Star Wars is an American epic space-opera multimedia franchise created by George Lucas." we have to be able to find that the surface forms like "America", "Star Wars", "George Lucas" maps to the respective DBpedia resources dbr:Star_Wars, dbr:George_Lucas and dbr:United_States.

In order to solve this problem, we have used DBPedia spotlight, where it takes a Wikipedia summary as an input, and returns the entities that exists in the text and along with their confidence scores.

For each entity that is extracted using DBpedia spotlight, we extract its related triples from the dataset of Mapping-based Objects. This finally helps in creating an aligned dataset which consists of 2.73M of DBpedia triples aligned with 250K Wikipedia summaries where we keep only the first 2 lined from them.

Using this aligned dataset, a model can be trained, but without a GPU and a good RAM, the training of a model was not possible. So for this research we picked up an already trained model and just tested it by providing it with triples and checking the generated output with the help of a grammar checking tool.

Chapter 5

Evaluation and Results

5.1 Experiments and Results

This section describes the experimental setup and the metric used to evaluate the systems. Also what values were achieved using these metrics.

5.1.1 Query Construction System

In order to evaluate the system that creates SPARQL query from the NL question, we will be using the QALD benchmark dataset which was explained in the Introduction section. According to the QALD challenge 9 Ngomo (2018), the evaluation metric used is F1 score. The formula for F1 score is

$$F1 = (2 * precision * recall)/(precision + recall)$$

where

$$Precision = TruePositives/(TruePositives + FalsePositives)$$

and

$$Recall = TruePositives/(TruePositives + FalseNegatives)$$

In Ngomo (2018), the F1 score for gAnswer Zou et al. (2014)(the approach we implemented for this research) is **0.430** which is best amongst all the submissions.

This experiment was done on DBPedia Knowledge Graph, which contains 5.4 millions entities, over 100 million triples and 9700 predicates. Also the relation mention dictionary contained 3.8 million entity pairs and 350K textual patterns. But this size of mappings dictionary requires over 20 GB of main memory. So in order to overcome this limitation, we reduced the size of these dictionaries so that they could take around 5 GB of RAM (as my system has 8 GB of RAM), which also reduced the performance and hence the F1 score got lower for the same dataset of QALD which came out to be **0.315** by using the following table.

Total questions	408
Queries created	264
Correct queries	106

Table 5.1: Evaluation of gAnswer approach on QALD challenge 9 dataset

When the synonyms approach along with the SPARQL to NL approach (combined called Query refine) was infused in the above approach of query translation, the number of questions that we converted to SPARQL got increased marginally and the F1 score cam out be **0.327** which is better than the results when this approach is not applied.

Total questions	408
Queries created	276
Correct queries	112

Table 5.2: Evaluation of gAnswer approach on QALD challenge 9 dataset along with the additional steps of synonyms added.

One of the questions that could not be converted into a query from QALD dataset was "Who is the mayor of New York?". But after adding the query refine step, we were able to convert it into SPARQL and get the correct answer.

In Diefenbach et al. (2017), a user study is shown, where the requirement of a system which can covert SPARQL query to NL representation is shown. This study states that, users are indeed interested in such type of system as it helps in reconfirming the generated SPARQL query, and also it can help the user to refine their own question phrase. Though our system does not allow any human interaction during conversions and translations, but we found this study very useful and due to this study we decided to integrate this system into our end-to-end QA system.

5.1.2 Text Generation from Triples

As the textual generation system demands to train a neural network model, it requires a huge dataset as well some advanced computational resources for training the model. So for this research we used an already trained model and used it to generate the summaries.

After we have the answer for a question in the form of RDF triples, we convert them using the pre-trained model. Once we have the generated summary, we check it using an online grammar checking tool called Grammarly and use its open source API. This tool helps us evaluate the summaries, as Grammarly provides a score out of 100 for a pirce of text.

So for the evaluation of the summaries, we set a threshold at 60, that a text summary has to score above 60 in order to be considered as a correct natural language representation. Another scenario could be that there are grammatical errors in the text, in that case no score is given. So in total, out of the 112 correct answer from the previous stage, 109 of them got converted correctly into a human readable summary. Fig 4.2 shows the result of the final application, where for a given question, we provide a SPARQL query, that query's natural language representation, and finally the textual answer of the triples fetched after hitting that query on DBpedia's Knowledge Base.

5.2 Discussion

The results presented above were calculated in order to provide answers to the research questions defined in Section 1.4, and re-stated below:

Question 1: Is it feasible to create a Question Answering agent over Knowledge Base Graphs that takes a Natural Language question as input and returns a human readable natural language summary/answer for that question?

Answer 1: From the above results and taking the outputs of the system into consideration, we can day that we have created a Question Answering system over Knowledge Base Graphs, that is capable of taking a natural language question from the user and return and human readable answer for that question.

Question 2: Can we enhance the accuracy of the existing state-of-the art Q/A agent that converts a natural language question into SPARQL query?

Answer 2: In section 5.1.1, we can see that after adding the synonyms approach to the existing system, we were able to bring up the F! score from 0.315 to 0.327. This was achieved because the system is capable of converting more natural language questions into SPARQL queries.

Hence we can say that we have answered both the research questions with positive results.

5.3 Limitations

1. In the query construction stage, for a better performance and accuracy, we require a system with approx. 20 GB of RAM which I could not get. So for this research we have used a smaller dataset that requires 5 GB RAM, but this affects the accuracy while generating SPARQL query.

2. While generating the natural language summary from rdf triples, we require a system with GPU, and this stage requires a lot of training data, which requires high computational resources. This requirement was a hindrance for achieving better performance.

3. This Question Answering system only supports English language, whereas there are numerous other query construction literatures that support multilingual questions.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

This research discussed the use of a graph based approach in order to convert a natural language question into a SPARQL query. After cleaning the question phrase by removing stop words and tokenizing it, a dependency tree is created using POS tagging. Then it is followed by entity and relations extraction which are used to create a super semantic query graph. Then this graph is matched over a Knowledge Base, and after we get matches for resources and predicates, we convert it into a SPARQL query. There could be instances where the system could not generate a query, in that case we tried the synonyms approach and were able to get answers for some of the extra questions. This part of the system was evaluated using the QALD dataset that contains 408 questions.

Then we move to the conversion of triples to NL step, where we use a pre-trained neural network model to generate natural language summaries. This model uses an encoder-decoder architecture which uses the sequence-to-sequence approach. In this approach the summary is built token by token. It calculates the conditional probability of the next token based on the next token and the previously select tokens and pick the current token with the best probability.

There were 3 different type of systems combined in the form of a pipeline to create one complete end-to-end system for solving the purpose of this research. Finally through this research we are able to answer out research question with positive results.

6.2 Future Work

As this system is not multilingual, in future this end-to-end natural language Question Answering system over Knowledge Bases can be made to be used for multiple languages, where a user can ask question in their native language and get the answer in the same language.

Also in this system, in order to improve the accuracy of the query translation system, we tried the synonyms approach. But other approaches can also be tried which are more robust and can improve the accuracy even further.

Bibliography

Aggarwal, N. and Buitelaar, P. (2012). A system description of natural language query over dbpedia. In *ILD@ ESWC*, pages 96–99.

Cabrio, E., Cojan, J., Aprosio, A. P., Magnini, B., Lavelli, A., and Gandon, F. (2012). Qakis: an open domain qa system based on relational patterns. In *International Semantic Web Conference, ISWC 2012*.

Chisholm, A. and Hachey, B. (2015). Entity disambiguation with web links. *Transactions of the Association for Computational Linguistics*, 3:145–156.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Daiber, J., Jakob, M., Hokamp, C., and Mendes, P. N. (2013). Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th international conference on semantic systems*, pages 121–124.

Diefenbach, D., Dridi, Y., Singh, K., and Maret, P. (2017). Sparqltouser: Did the question answering system understand me? In *ISWC 2017*.

Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 1535–1545.

Freitas, A. and Curry, E. (2014). Natural language queries over heterogeneous linked data graphs: A distributional-compositional semantics approach. In *Proceedings of the 19th international conference on Intelligent User Interfaces*, pages 279–288.

Gao, H., Wu, L., Hu, P., and Xu, F. (2021). Rdf-to-text generation with graph-augmented structural neural encoders. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3030–3036.

Green, N. (2006). Generation of biomedical arguments for lay readers. In *Proceedings of the Fourth International Natural Language Generation Conference*, pages 114–121.

He, S., Zhang, Y., Liu, K., and Zhao, J. (2014). Casia@ v2: A mln-based question answering system over linked data. In *CLEF (Working Notes)*, pages 1249–1259.

Lapalme, G. (2020). Rdfjsrealb: a symbolic approach for generating text from rdf triples. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 144–153.

Lopez, V., Fernández, M., Motta, E., and Stieler, N. (2012). Poweraqua: Supporting users in querying and exploring the semantic web. *Semantic web*, 3(3):249–265.

Lopez, V., Unger, C., Cimiano, P., and Motta, E. (2013). Evaluating question answering over linked data. *Journal of Web Semantics*, 21:3–13.

Ngomo, N. (2018). 9th challenge on question answering over linked data (qald-9). *language*, 7(1):58–64.

Ngonga Ngomo, A.-C., Bühmann, L., Unger, C., Lehmann, J., and Gerber, D. (2013). Sorry, i don't speak sparql: translating sparql queries into natural language. In *Proceedings of the 22nd international conference on World Wide Web*, pages 977–988.

Park, S., Shim, H., and Lee, G. G. (2014). Isoft at qald-4: Semantic similarity-based question answering system over linked data. In *Clef (Working Notes)*, pages 1236–1248.

Reiter, E., Sripada, S., Hunter, J., Yu, J., and Davy, I. (2005). Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, 167(1-2):137–169.

Shekarpour, S., Endris, K. M., Jaya Kumar, A., Lukovnikov, D., Singh, K., Thakkar, H., and Lange, C. (2016). Question answering on linked data: Challenges and future directions. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 693–698.

Spitkovsky, V. I. and Chang, A. X. (2012). A cross-lingual dictionary for english wikipedia concepts.

Turner, R., Sripada, S., and Reiter, E. (2009). Generating approximate geographic descriptions. In *Empirical methods in natural language generation*, pages 121–140. Springer.

Vougiouklis, P., Elsahar, H., Kaffee, L.-A., Gravier, C., Laforest, F., Hare, J., and Simperl, E. (2018). Neural wikipedian: Generating textual summaries from knowledge base triples. *Journal of Web Semantics*, 52:1–15.

Xu, K., Zhang, S., Feng, Y., and Zhao, D. (2014). Answering natural language questions via phrasal semantic parsing. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 333–344. Springer.

Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., and Weikum, G. (2012). Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 379–390.

Yosef, M. A., Hoffart, J., Bordino, I., Spaniol, M., and Weikum, G. (2011). Aida: An online tool for accurate disambiguation of named entities in text and tables. *Proceedings of the VLDB Endowment*, 4(12):1450–1453.

Zhu, Y., Wan, J., Zhou, Z., Chen, L., Qiu, L., Zhang, W., Jiang, X., and Yu, Y. (2019). Triple-to-text: converting rdf triples into high-quality natural languages via optimizing an inverse kl divergence. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 455–464.

Zou, L., Huang, R., Wang, H., Yu, J. X., He, W., and Zhao, D. (2014). Natural language question answering over rdf: a graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 313–324.