Technical Note

Natural Language Processing-Based Querying Heterogeneous Data Sources Using Integrated Ontology

Swathi S *

Posted Date: 4 December 2023

doi: 10.20944/preprints202312.0228.v1

Keywords: Natural language processing, Ontology, Knowledge graph, Heterogeneous data sources

Natural Language Processing-Based Querying Heterogeneous Data Sources Using Integrated Ontology

Swathi S [1,*]

[1] PSG College of Technology; swathissara@gmail.com
* Correspondence: swathissara@gmail.com

Abstract: It is essential to deal with data scattered among heterogeneous information sources, which can be structured, semi-structured, or unstructured, to give consumers a cohesive perspective of the data. Information gathering is challenging as a result, and one of the major causes of this is that data sources are developed to support specific applications. A method to simplify this procedure is to use ontology representation as an intermediate step. An ontology represents a knowledge structure that reasonably reflects the real world's complexity and is ideally used in many industries today. This research aims to enhance the functionality of a global ontology system by implementing and integrating a natural language query system. Leveraging NLP-based approaches, we constructed SPARQL queries for our ontologies to facilitate natural language translation into a well-structured query format. A dedicated query component was developed to transform natural language questions into SPARQL queries, with the selection of the optimal query graph to generate the final queries. In the subsequent phase, we explored the integration of distinct sentence encoders to improve latent sentence representations during query construction. The generated SPARQL queries were executed on the global ontology output and then translated into source-specific questions. This innovative approach enables unified access to heterogeneous data sources through a user-friendly natural language querying interface.

Keywords: Natural language processing, Ontology, Knowledge graph, Heterogeneous data sources

1. Introduction

Ontology is a philosophical discipline about life and nature. In social sciences and computer science, ontology refers to the representation of certain knowledge or concepts. The purpose of this is to identify elements, categories, and their relationships to better understand the field. It is a language that provides efficient communication and information sharing between people or systems working in the same field. It can also provide design information that can be processed by machines, for example, using artificial intelligence. To create an ontology, you must first define the categories or classes and their properties. Next, the relationship between these classes should be specified. The results of ontologies can be represented in different formats, such as RDF or OWL, and can be used to support many applications including search engines, experts, and knowledge management. We know this from the business example, that the semantic signature is only done in the context of ontology classes or data attributes. Combining this knowledge is the infinite relationship of events from one ontology class to another, the ability of events to represent value, the choice of a class ontology for others, and the integration of events. For example, transfer values allow switching from one ontology class to another. Each must have a relationship, such as equality or relationship, that limits the value (data volume) of the attribute and limits its value. Each defined location can be defined as a direct share type, multiple data share type, or transfer type.

Semantic Data Integration offers solutions that go beyond traditional business application integration. It publishes and exchanges data using a standard architecture: a data-centric architecture based on RDF. A heterogeneous knowledge organisation (structured, semi-structured, and unstructured) can be represented, stored, and accessed using this framework. Because the data structure is represented by connections in the data, it is not constrained to the structure mandated by the data, and no data evolution occurs. When data changes, the database connections change and are reflected in the database. QA (Knowledge Graph) clearly demonstrates data interpretation by analysing content, object relationships, and groups of items.

We used the following databases for semantic integration: MongoDB is a resource-enabled database. MongoDB is classified as a NoSQL database program that uses JSON-like data and an arbitrary schema. MongoDB, MongoDB Inc. Developed by and licensed under the Server Side Public License, which some distributions have decided are not free. Neo4j is the world's most popular open source graphical database built with Java technology. It is extremely scalable and schema-free (NoSQL). Neo4j is a popular image database. Other graph databases include Orient DB, Hyper GraphDB, Graph Base, Infinite Graph, and Allegro Graph. For many applications, it is better than a row-column database or a plain JSON data database, mainly for some big data and analytics applications.

In fact, NLP (Natural Language Processing) involves processing text to extract important information and search or store relevant information. To make an NLP-based query, you can follow these general steps: ***Define your query***: Identify the specific data or information that will be provided by the NLP questions. ***Data collection and preparation***: collection of relevant data and preparation for NLP analysis. This will include steps such as clearing data, clearing latency, and adjusting data. ***Train or use pre-trained NLP models***: You can use pre-trained NLP models such as BERT, GPT-3, or custom models trained on your specific data to process notes and provide relevant information. ***Use queries with NLP models***: Feed queries into the NLP model and let it process the text and provide relevant information. ***Evaluate Results***: Evaluate the results to make sure they match and are relevant to your question. You may need to adjust the issue or NLP model to get more accurate results. ***Promotional Information***: Display relevant information, such as descriptions, images, or related product names in a helpful and understandable way in a simple way during the recovery period. NLP-based querying involves a combination of language processing techniques, machine learning, and data analysis to extract meaningful content from unstructured text.

Integrating disparate data using a unified approach can be a difficult process, but here are some steps: ***Check different data***: Check the information we need for our information. ***Create a local ontology***: Build a local ontology using the heterogeneous information we need. **Use testing, training, and data prediction models to find consistency**: logistic regression, XGBoost, etc. of our local ontologies. Perform all testing, training, and learning models, such as aspects, and evaluate the similarity measure. ***Create a global ontology***: Create a global ontology from the semantic integration of local ontologies. ***Data query***: load data into the integrated ontology and users can use the content and relationships in the ontology to query the data. It is important to note that using ontology to integrate disparate data requires expertise in data integration and ontology design. Ontology query systems have many advantages, for example, ***Enhanced truth discovery***: Ontology-based systems help improve truth discovery by using ontology context and relationships to better understand the context, language, and context of user questions. This will help achieve better results and reduce negative and unwanted effects. ***Scalability***: Ontology-based systems can be easily extended to manage large data sets because ontologies provide a framework for organizing and managing complex data. ***Consistency***: Ontologies provide consistent content and reference standards, ensuring that all data and queries are expressed in the same language. This will help reduce confusion and confusion in data analysis and decision making. ***Interoperability***: Ontologies can be used to facilitate information sharing and interoperability between different systems and applications. This will help integrate information from different sources and improve collaboration between different stakeholders. ***Adaptability***: Ontology-based systems are adaptable and can be easily adapted or modified to reflect changes in the domain. This will help ensure that the system remains relevant and

 doi:10.20944/preprints202312.0228.v1

efficient over time. In general, ontology based systems provide powerful tools for data management and analysis, helping organizations make smarter decisions and extract more value from data.

Ontology in health care refers to the creation of models and structures related to health knowledge and ideas. It involves creating a common language that can be used to organize and store health-related information and data. Ontologies can be used to represent information about various medical fields, such as diseases, symptoms, drugs, procedures, and patient information. They can also be used to collect data from a variety of sources, such as electronic medical records, medical records, and clinical trials. One of the main benefits of using ontology in healthcare is that it facilitates health interactions and information exchange among stakeholders.

2. Related Work

The paper [1] aims to review current research papers that address the issue of translating competency questions from natural language into SPARQL queries in the context of the web. This essay makes a recommendation for a classification that takes into account a number of factors influenced by usability and selection criteria. The classification suggested by this study is based on a collection of traits driven by selection and usage incentives. We were able to develop a general strategy with five main activities using the current study's data on frequently observed translation processing phases. finished with the three mapping functions: First, a simple label matching function is carried out. The second test for mapping is the WordNet synonym function, which is used if the first one fails. The user's requested synonym is located inside the collection using WordNet. The mapping of the languages is based only on syntactic similarities. However, if this presumption is false, semantic ambiguities will significantly worsen the performance of the systems that are created. Many of the explored strategies address simple problems. A few of them concentrate on the issue of translating complicated compound questions. These techniques often rely on powerful logical and mathematical principles. The work on these subjects calls for more effort. Knowledge graphs are another method presented in [2] that is beneficial for searching large amounts of material. These knowledge networks, however, are frequently huge and difficult for end users to access since they require a specialized understanding of query languages such as SPARQL. They proposed a new QA method for turning natural language queries into SPARQL queries. The main idea is to use ensemble machine learning techniques and Tree-LSTM-based neural network models to automatically learn and translate a natural language inquiry into a SPARQL query by breaking the translation process down into five smaller, more manageable sub-tasks. The 7th Question Answering over Linked Data Challenge (QALD-7) and the Large-Scale Complex Question Answering Dataset are two well-known benchmarks used to empirically assess the performance of their proposed QA system. It needs neither time-consuming feature engineering nor a set of heuristic rules for translating a natural language query into a template for a SPARQL query. Because the suggested system's query generation model and question type categorization methodology don't require any domain-specific expertise, it can be easily deployed to previously unexplored topics. The diversity of training data is currently limited because there are only three different types of questions in the training datasets that are available. to increase the amount and quality of the training dataset, did not use complicated queries containing the stated operators.

In paper [3] To accurately infer non-taxonomic relations from a domain, it is required to understand its semantics. It is necessary to thoroughly investigate machine learning classifiers for feature classification. We contrasted Support Vector Machines, Artificial Neural Networks (ANN), and Bay's classifiers for feature classification. Experiments show that knowledge extraction from unstructured text input is best accomplished using the SVM machine learning classifier. 95%of the new pages were accurately extracted. 5 additional criteria were added to increase accuracy, bringing it up to 99%. This demonstrates that there is a finite and controllable amount of rules and patterns required to extract reliable information from various websites. These designs can be manually created in a respectable length of time. The updated ontology can then be used to answer semantic queries that would otherwise be intractable. As was already said, they have methodologies that have been

 doi:10.20944/preprints202312.0228.v1

published elsewhere that offer a simple form-based interface and generate semantic queries without the requirement for new rule creation.

A new method was introduced in [4] In order to allow data sharing, exchanging, and integration 14 amongst multiple data sources, ontology-based data access (OBDA) provides semantic access to a number of heterogeneous data sources. The OBDA method restricts end users' ability to specify their requests because formal query languages, such as SPARQL, are typically employed to convey user questions. A layer that receives user requests in their native tongue and converts them into one of these formal languages is now required to address this issue. To do this, they developed a novel, interactive system that directs the user while translating. The suggested method takes advantage of both the semantic data included in the domain ontology and the capabilities of natural language processing.

The suggested strategy also takes user participation into account while translating. We put the suggested strategy into practice and validated it using a query benchmark that measures query accuracy and efficiency in order to show its efficacy. In order to provide effective interaction and relationships between each other, this system primarily designed two interfaces: user and processing interfaces. Although it wasn't done, enhancing the results by revealing the semantic relationships between terms is something they may do in the future. Modifiers can be used to supplement and enhance the suggested translator.

In this paper [5] Ontologies are used in the medical industry to represent medical words. They make it possible for systems that use terms that are lexically different but semantically related to communicating more effectively. It improves the sharing and reuse of patient and medical data. Additionally, the percentage of people who support knowledge and data integration has increased. The Covid-19 ontology with 158 classes and a maximum depth of 12 levels was used in the suggested case study. The Semantic Web Rule Language and Simple Protocol and RDF Query Language (SPARQL) are used to retrieve information from the knowledgebase Semantic Web Rule Language (SWRL). Using a template based on natural language, simple queries can be created. The information at hand can also be used to develop new knowledge. Queries can be given using SPARQL and Semantic Query-Enhanced Web Rule Language (SQWRL) from the proposed ontology retrieval system, and rules can be generated using SWRL. In such scenarios, user queries are simplified by leveraging enhanced parser output and extracted concepts, instances, and relations. The knowledge base is then queried using refined queries. Retrieval utilizing NLP-based technology is recommended to manage user questions with limited knowledge of ontology and query language syntax.

The system described in [6] accepts a complete question in English, determines the type of query that must be built, and finds the triples from the question that corresponds to the query. The first step in the methodology described in this paper is a part-of-speech analysis and dependency parsing of the linguistic content of the inquiry. For a classification task, these are quantified and provided into a recursive neural network model. The outcomes of the entity recognition and classification activities are combined to generate a complete SPARQL query. The system outlined in the paper builds an end-to-end question answering system for RDF data in order to offer an accurate SPARQL query. It does this by utilizing a variety of machine learning and natural language processing techniques. Template classification in use It works well to identify the question template for template classification using the tree-LSTM technique. The absence of ontology recognition is one of the key drawbacks. The system can only match terms in the normal Wikidata ontology due to the use of third-party entity-linking libraries. Phrase-matching needs to be customized using different datasets and attributes.

The use of neural machine translation (NMT) models to translate from natural English to the structured query language SPARQL is evaluated in [7]. The goal of NMT is to automatically translate across different NLPs. The architecture of the encoder-decoder is a well-known illustration of NMT. High-quantity, high-quality datasets are stressed in the paper, which also notes that the Convolutional Neural Network (CNN)-based architecture model produces the greatest outcomes. Each of the models was trained and evaluated on three different datasets to guarantee the word order of the generated SPARQL queries. A simple accuracy metric, BLEU scores, and perplexity were all used in the evaluation process.

 doi:10.20944/preprints202312.0228.v1

The purpose of [8] is to examine contemporary natural language querying frameworks for databases. Frameworks for connecting natural languages to databases have been around for four decades, and numerous initiatives have been launched to help end users. Small-scale databases were the focus of the first tools, which meant that systems built around them were not suitable for commercial usage. With the development of new commercial natural language to database querying frameworks, support for various databases was added. The majority of the surveys that are now available are obsolete, but relevant research and surveys have been discovered in the literature that concentrate specifically on the natural language to database querying frameworks.

Interoperability is offered through the NLP Interchange format, which operates in the semantic web domain. A query is built in a very little period of time. One disadvantage of employing supervised learning techniques is the need for large training datasets with human labeling, which eventually adds to time, expense, and labor. Dependence on statistical data without understanding the context's importance has the disadvantage of predominating.

A brand-new user similarity metric is presented [9]. It can still determine user similarity even if two users do not have any co-rated items, unlike the traditional similarity measures (such as COS and PC). The suggested user similarity is therefore more suited for sparse data. In order to design item similarity according to the probability distribution of ratings, the KL divergence approach is introduced in the signal processing and information theory sector. The requirement that at least one user must rate the two things simultaneously is removed. The KLCF being presented has a 3% advantage. Additionally, the KLCF improves accuracy (F1 measure) by over 10%. In this study, KL divergence was employed to reduce the amount of information lost when approximating a distribution.

Similarly in [10], document-oriented NoSQL databases are used to implement a novel technique for OBDA. This approach combines an extensible, flexible access interface with an intermediate conceptual layer that can give access to many database management systems, unlike comparable attempts. As a proof-of-concept, they have built a MongoDB prototype using a real-world application domain as a case study. By avoiding a direct mapping to the database query language, the amount of work needed to establish and maintain the mapping is reduced. Unlike relational DBMSs, which specify queries in standard SQL language, every NoSQL DBMS has a distinct and flexible query language.

3. System Architecture

1. Query Type Classifier: As can be seen in Figure 1, the first and most important stage begins with the categorization of the question type. The NLP question will be obtained, and it will divide it into three types, namely List, Boolean, and Ask questions.

The type of responses that a specific question will demand are predicted by a machine learning classifier known as the Question Type Classifier. For instance, how many mammals are classified under the Chordate phylum? The fact that a count is required makes it a count issue. Such information is necessary in order for the Query Constructor to create the final SPARQL query with the correct aggregation function.

2. Query Graph Generator: In the following section, the Query Graph Generator accepts input in the form of mapped entities. The subcomponent Graphs Generator creates candidate query graphs. Because we only know the resources involved in our enquiry, we are unable to determine the suitable query graph that should be used to build the SPARQL query. This leaves us with little choice except to present every possible graph and then order them based on how closely they resemble the user's query. There are two steps in this section. The first step entails building a subgraph based on the knowledge base (KB) that serves as the foundation and contains all input resources and any connections that might exist between them. On the basis of this graph, the second stage creates each valid candidate query graph.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1

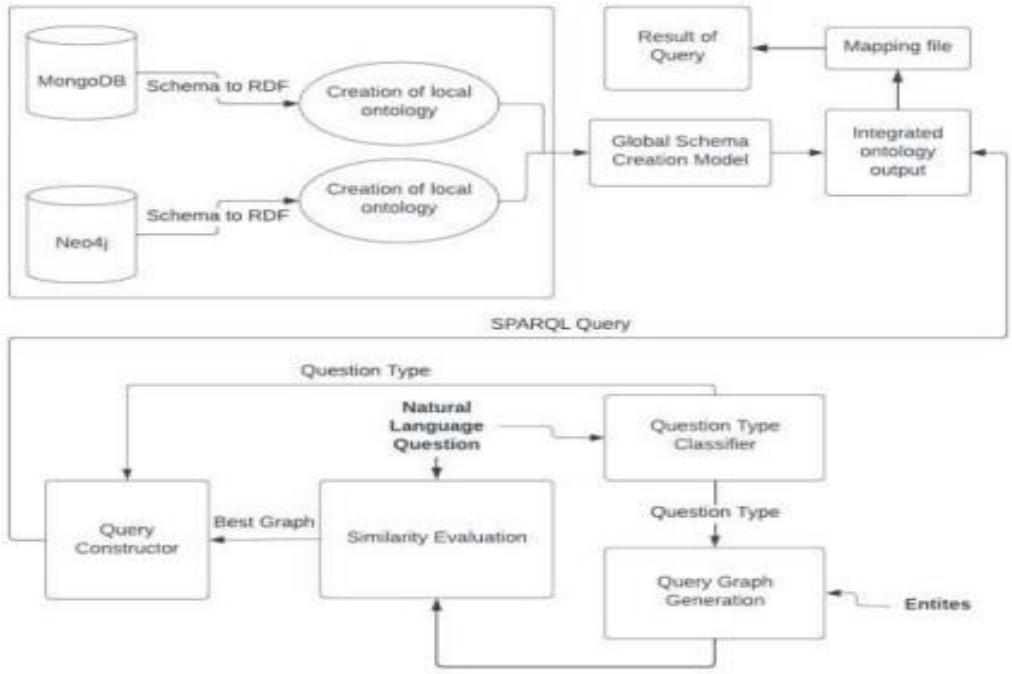

Figure 1. System Design

3. Evaluation of Similarity: The minimal subgraph result is now supplied as input for determining how similar the two are. Both the results of the Query graph generator and NLP questions are used as inputs for similarity evaluation. The similarity evaluator's job is to contrast each potential query graph with the natural language question supplied by the user. This component assesses each inquiry separately and generates the likelihood that the question will receive a suitable response. To find out how similar the sentence and the graph are to one another, we first compute their latent vectorial representations. Then, pairs of question-graph vectors are fed into a function of similarity, like the cosine similarity.

4. Query Constructor: To build the SPARQL query, the Query Constructor is the next step. The information provided by the Similarity Evaluator and Question Type Classifier is gathered by this sub-component. The most likely candidate inquiries are chosen after the candidate queries are ranked by likelihood. Additionally, SPARQL templates are utilized. There is a specific template for every type of query. The Query Constructor's job is to put everything together and provide a SPARQL query that responds to the user's query. A local ontology is built as part of the design's second phase using data from several repositories. Create an integrated global ontology from the observed alignments by using the entity alignment model, which employs semantic similarity measurements, to find entities in local ontologies that may be joined.

Finally, the integrated global ontology receives the SPARQL query that was produced by the Query Constructor as input.

3.1. Dataset Description

Recent research has focused on powering computers to search and answer questions in the survey. An important method among them is to change the questions from natural language to formal questions. Many datasets such as Web Questions [13], QALD [14], and LCQuAD [15] have been published previously to enhance the research. We used LC-QuAD 2.0 (Large-Scale Complex Question Answering Dataset) because it contains different questions. It comes with 30,000 duplicate queries, their definitions, and accompanying SPARQL queries. Both Wikidata [16] and DBpedia [17] 2018 infographics are compatible with LC-QuAD 2.0. It includes 21,258 unique and 1,310 unique relationships and 22 unique SPARQL query templates. We also use the conference dataset from OAEI [18]. The conference work consists of 16 ontologies of the same name (foundation conference). These ontologies are well suited for ontology-matching tasks as their environments are heterogeneous. This

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1

year it is supported by the MELT framework (see MELT assessment for OAEI 2021). A list of assessment methods can be foundin the next section. There are seven ontologies for communication: Sat, ConfTool, Edas, Ekaw, lasted, Sigkdd, and Sofsem. A number of ontologies deal with the organization of meetings. They were created in the OntoFarmproject. There are 16 ontologies. A number of ontologies deal with the organization of meetings. They were created in the OntoFarm project. There are 16 ontologies.

Who are the employees of Food and Drug Administration?

```
SPARQL DBpedia18: select distinct ?answer where { ?statement . ?statement . ?
statement ?answer. }
```

Who is the developer of Free Software Foundation?

```
SPARQL DBpedia18: select distinct ?answer where { ?statement ?answer. ?statement
. ?statement . }
```

Who is the translator of The Jungle Book?

```
SPARQL Wikidata : select distinct ?obj where { wd:Q189509 wdt:P655 ?obj . ?obj
wdt:P31 wd:Q5 }
```

```
SPARQL DBpedia18: select distinct ?obj where { ?statement . ?statement . ?
statement ?obj . ?obj }
```

Figure 2. Dataset Details

Four different datasets have been created and are stored in three different databases. This dataset includes the details of patient medical records, cardiovascular disease results, symptoms that each patient has, and the details of the doctor who treats the patient. Each dataset consists of 1000 records and the doctor dataset consists of 50 records.

1. Patient medical record dataset consists of patient ID, age and sex of the patient, cholesterol level, ECG result, maximum heart rate achieved, old peak = ST depression induced by exercise relative to rest, exercise-induced angina, fasting blood sugar, the slope of peak exercise ST segment. If the patient has some abnormalities the result is stored in the form of 1s and the normal result is stored in the form of 0s. This dataset has been stored in Neo4j.
2. Cardiovascular disease result dataset consists of Patient ID, detection of blocks, maximum heart rate, presence of cholesterol, and CVD result depicting the severity of result stored in Mongo DB.
3. Doctors' dataset consists of Doctor ID, Patient ID who is under that doctor's observation, the Mobile number and address of the doctor, the name of the hospital that the doctor visits, and clinic timing. This dataset is stored in MySQL.
4. Symptoms dataset consists of common and general symptoms of a person who is having a heart disease such as left arm pain, maximum heart rate, blood vomiting, dizziness, fatigue, and chest pain. They are stored in the form of Boolean values.

This dataset is stored in MySQL. The next step is the conversion of the .csv files into RDF and OWL format. The following figures represent the rdf and owl format of the 4 datasets.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1


```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
>
  <owl:Class rdf:about="http://example.org/Patient_ID">
    <rdfs:subClassOf rdf:resource="http://example.org/patient_record_neo4j"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/Age">
    <rdfs:subClassOf rdf:resource="http://example.org/patient_record_neo4j"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/sex">
    <rdfs:subClassOf rdf:resource="http://example.org/patient_record_neo4j"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/cp">
    <rdfs:subClassOf rdf:resource="http://example.org/patient_record_neo4j"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/trtbps">
    <rdfs:subClassOf rdf:resource="http://example.org/patient_record_neo4j"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/chol">
    <rdfs:subClassOf rdf:resource="http://example.org/patient_record_neo4j"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/fbs">
    <rdfs:subClassOf rdf:resource="http://example.org/patient_record_neo4j"/>
  </owl:Class>
```

Figure 3. OWL format of Patient Records

```
<?xml version="1.0"?>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"/>
    <Prefix name="xml"
IRI="http://www.w3.org/XML/1998/namespace"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-
schema#"/>
    <Declaration>
        <Class IRI="http://example.org/Address"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://example.org/ClinicTimings"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://example.org/HospitalVisiting"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://example.org/MobileNo."/>
    </Declaration>
    <Declaration>
        <Class IRI="http://example.org/NameoftheAMA"/>
    </Declaration>
```

Figure 4. OWL format of Doctor Details

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1


```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
>
  <owl:Class rdf:about="http://example.org/Patient_facing_issues">
    <rdfs:subClassOf rdf:resource="http://example.org/symptoms_mysql"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/Max_heartrate">
    <rdfs:subClassOf rdf:resource="http://example.org/symptoms_mysql"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/oldpeak">
    <rdfs:subClassOf rdf:resource="http://example.org/symptoms_mysql"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/left_arm_ain">
    <rdfs:subClassOf rdf:resource="http://example.org/symptoms_mysql"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/dizziness">
    <rdfs:subClassOf rdf:resource="http://example.org/symptoms_mysql"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/fatigue">
    <rdfs:subClassOf rdf:resource="http://example.org/symptoms_mysql"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/chest_pain">
    <rdfs:subClassOf rdf:resource="http://example.org/symptoms_mysql"/>
```

Figure 5. OWL format of Symptoms

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
  <owl:Class rdf:about="http://example.org/Patient_id">
    <rdfs:subClassOf rdf:resource="http://example.org/cvd_mongo"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/cvd_mongo">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/
    <rdfs:subClassOf rdf:resource="http://example.org/cvd_mongo"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/detection_of_blocks">
    <rdfs:subClassOf rdf:resource="http://example.org/cvd_mongo"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/max_heart_rate">
    <rdfs:subClassOf rdf:resource="http://example.org/cvd_mongo"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/coronaryangiogram">
    <rdfs:subClassOf rdf:resource="http://example.org/cvd_mongo"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/presenceofcholesterol">
    <rdfs:subClassOf rdf:resource="http://example.org/cvd_mongo"/>
  </owl:Class>
  <owl:Class rdf:about="http://example.org/cvd_result">
```

Figure 6. OWL format of CVD Test Results

4. Proposed Solution

4.1. Dataset Description

To cater to this aim, we analyze the correctness of three separate classes. Query type classification is the process of classifying queries on three wide spectrums of Boolean type, count type, and list type. The results, however, are independent of the entity/relation linking module because it provides no new input. To train the model and discover the best parameter values, we use 10-fold cross-validation on 50% of the dataset.

The performance comparison with 2 different classifiers and where feature identification accuracies are shown.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1

Logistic Regression gives us an accuracy of 98.50%. Hence, we can conclude that Logistic Regression is a better classifier for our use case.

4.2. Query Graph Generation

The objective of the Query graph generation component is to generate a SPARQL query from a natural language question. The "Question Type Classifier" component above is tasked with identifying whether a question belongs to the categories of 'List', 'Count', or 'Boolean'. This component is responsible for generating the 'SELECT' clause in the SPARQL query. The subsequent step in constructing the SPARQL query involves determining the 'WHERE' clause, which is the objective of the "Query Generation" component. This component outputs a set of candidate query graphs, which we rank later in the process, according to how similar they are to the user's question. In general, the process of query generation consists of two steps:

a) *Subgraph Construction*: Candidate queries are constructed by enumerating the valid walks in the Knowledge Graph. Considering the size of established Knowledge Graphs (KGs) like DBpedia or Freebase, the process of listing all possible valid walks can be extremely time-consuming. As a result, we narrow our focus to the subgraph that encompasses all the identified entities and relations. Within this subgraph, we can efficiently enumerate the potential candidate walks, which can subsequently be mapped to SPARQL queries. Traversing the subgraph significantly decreases the computation time compared to other methods. By considering the entire knowledge graph instead, there would be precision and execution time performance drawbacks.

```
[14] from sklearn.svm import SVC
     classifier = SVC(kernel = 'linear', random_state = 0)
     classifier.fit(X_train, y_train)

     SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
         decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
         max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
         verbose=False)

[15] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

[16] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)

     [[103   5]
      [  2  61]]
     0.9590643274853801

     from sklearn.model_selection import cross_val_score
     accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
     print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
     print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))

     Accuracy: 97.49 %
```

Figure 7. Model accuracy of Support Vector Machine.

The construction of the subgraph is done using Algorithm 1 in Figure 7. We start by establishing an empty subgraph G, which is subsequently enriched with the entities (E) as its nodes. Next, we augment the nodes with edges that correspond to the relations (R), if such connections exist in the Knowledge Graph (KG). If a relation connects two entities of E, the algorithm only adds that relation as an edge. But if a predicate connects a node of E and another node e not included in E, then the algorithm adds both the relation and the node e. Such nodes are called unbound nodes. The second part of the algorithm expands G by including nodes that are at a two-hop distance from the entities E.

b) Candidate Query Generation: The subgraph has been constructed encompassing all the entities and relations mentioned in the question. The candidate queries are valid walks (Definition 2) in the subgraph, considering each unbound node in the subgraph as a potential answer node. Finding the valid walks, provides the candidate queries.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1

Figure 8 shows the candidate queries generated for the question "Which comic characters are painted by Bill Finger?".

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)

[9] y_pred = classifier.predict(X_test)
    print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

[10] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)

     [[107   1]
      [  3  60]]
     0.9766081871345029

     from sklearn.model_selection import cross_val_score
     accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
     print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
     print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))

     Accuracy: 98.50 %
```

Figure 8. Model accuracy of Logistic regression classifier

```
function graphConstruction(E, P, K):
    G = createEmptyGraph()  // Initialize G as an empty graph
    addNodes(G, E)          // Add E in G as nodes
    for each e in E and p in P do
        if (e, p, ?) is in K then
            addEdge(G, (e, p, ?)) // Add (e, p, ?) to G
        else if (?, p, e) is in K then
            addEdge(G, (?, p, e)) // Add (?, p, e) to G
        end if
    end for
    for each (e1, p, e2) do
        for each p' in P such that p != p' do
            if (e2, p', ?) is in K then
                addEdge(G, (e2, p', ?)) // Add (e2, p', ?) to G
            else if (?, p', e2) is in K then
                addEdge(G, (?, p', e2)) // Add (?, p', e2) to G
            else if (e1, p', ?) is in K then
                addEdge(G, (e1, p', ?)) // Add (e1, p', ?) to G
            else if (?, p', e1) is in K then
                addEdge(G, (?, p', e1)) // Add (?, p', e1) to G
            end if
        end for
    end for
    return G
end function
```

Figure 9. Algorithm for minimal subgraph construction

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1

Definition 1. (Walk) *A walk in a graph* $G = (V, E)$ *is a finite sequence of vertices and edges that alternately occur in the sequence, starting and ending with a vertex. Formally, a walk* W *in* G *is represented as* $W = (v_0, e_1, v_1, e_2, v_2, \ldots, e_{k-1}, e_k)$*, where* $v_i \in V$ *for* $0 \leq i \leq k$ *and* $e_i \in E$ *for* $1 \leq i \leq k$*.*

Definition 2. (Valid Walk) *A walk* W *(Definition 1) is valid with respect to a set of entities* E *and relations,* R*, if and only if it contains all of them,*

$$\boldsymbol{i.e.\ \forall e \in E: e \in W\ and\ \forall r \in R: r \in W}$$

A node e *with* $\boldsymbol{e \in W}$ *and* $\boldsymbol{e \notin E}$ *is bounded*

4.2. Similarity Evaluator

To choose the best query graph to employ for creating our SPARQL queries, we rank the candidate query graphs produced during the query generation phase. The semantic similarity between the candidate query and the NLP question are found using the similarity evaluator component and this semantic similarity is used to determine the ranking of the candidate queries.

The steps to identify the semantic similarity between the NLP question and the candidate queries are outlined below:

a) Preprocessing: Clean and preprocess the question and the candidate queries according to the encoding model used. While most models used for sentence encoding accept sequential input like Recurrent Neural Networks (RNN), and Long Short Term Memory (LSTM), encoders like Tree-LSTM are designed to receive trees. Therefore, dependency parse trees are constructed in such cases to give input in the appropriate format. A dependency parse tree is a tree where each node contains a vector representation of a word or phrase in the sentence and edges between nodes represent dependency relations. It highlights the syntactical structure of a sentence according to formal grammar, by exposing the relationships between words or sub-phrases.

b) Encoding the question: Once we have the input in the appropriate format for the encoding model, we then use them to encode the syntactic structure of the question into a vector representation.

c) Encode the candidate query: The candidate queries are also encoded using the same encoding model to get its vector representation. This vector representation reflects the semantic meaning of the candidate query.

d) Calculate the similarity: Appropriate similarity measures, such as cosine similarity, Jaccard similarity, orother distance metrics, to compare the encoded representations of the sentences are then applied to the two vector representations. The output of these similarity measures quantifies the similarity between the question and the candidate query.

The best query graph for SPARQL query generation can then be found by ranking candidate queries according to the estimated similarity score.

The solution explores two different encoding models: Manhattan-LSTM and Tree-LSTM. The main difference between the two is the kind of input they take. While Manhattan-LSTM uses LSTM networks that handle sequential information, Tree-LSTMs are specifically designed to accept tree-structured data, making it suitable for encoding the semantic and hierarchical relationships in sentences.

LSTM: Long Short-Term Memory (LSTM) [19] is a type of recurrent neural network (RNN) that is used for processing sequential data. LSTMs are designed to overcome the vanishing gradient problem in traditional RNNs, which occurs when gradients become too small and cause the network to forget information from earlier time steps.

The LSTM architecture includes three gates: the input gate, the forget gate, and the output gate. These gates control the flow of information into and out of the memory cell, allowing the network to selectively remember or forget information as needed.

The input gate is responsible for deciding which information from the current time step should be added to the memory cell. It takes as input the current input vector x_t and applies a sigmoid $t-1$ activation function to produce an "input gate activation" vector i_t that determines which elements of the input should be included in the memory cell:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1

$$i_t = sigmoid(W_i[h_{t-1}, x_t] + b_i)$$

The forget gate determines which information should be removed from the memory cell. It takes as input the same vectors as the input gate and produces a "forget gate activation" vector (f_t) that determines which elements of the memory cell should be forgotten:

$$f_t = sigmoid\big(W_f[h_{t-1}, x_t] + b_f\big)$$

The output gate decides which information from the memory cell should be output at the current time step. It takes as input the current input vector, the previous hidden state, and the current cell state, and produces an "output gate activation" vector (o_t) that determines which elements of the memory cell should be included in the output:

$$O_t = sigmoid(W_0[h_{t-1}, x_t] + b_0)$$

Tree-LSTM: The Tree-LSTM structure was first introduced by Kai et al. in an ACL 2015 paper: Improved Semantic Representations from Tree-Structured Long Short-Term Memory Networks. Tree-LSTM is a tree-structured network topology generalization of LSTMs. In two tasks, Tree-LSTMs outperform all existing systems and strong LSTM baselines: predicting the semantic relatedness of two sentences and sentiment categorization. The LSTM designs discussed in the preceding section have the constraint of only allowing strictly sequential information propagation. Here, the Child-Sum Tree-LSTM and the N-ary Tree-LSTM are two natural extensions of the fundamental LSTM architecture. Both variations enable richer network topologies in which each LSTM unit can incorporate data from many child units. Each Tree-LSTM unit (indexed by j) has input and output gates ij and oj, a memory cell cj, and a hidden state hj, just like regular LSTM units. The tree-LSTM units differ from ordinary LSTM units in that gating vectors and memory cell updates are dependent on the states of potentially many child units.

Also, rather than a single forget gate, the Tree-LSTM unit has one forget gate f_{jk} for each child k. This enables the Tree-LSTM unit to incorporate information from each child selectively. A Tree-LSTM model, for example, can learn to highlight semantic heads in a semantic relatedness task or to preserve the representation of sentiment-rich children in sentiment classification. Each Tree-LSTM unit, like the normal LSTM, accepts an input vector x_j. Each x_j in our apps is a vector representation of a word in a phrase. The input word at each node is determined by the network's tree structure.

The choice of similarity measure also affects the accuracy of the model.

5. Experiment and Results

For experimentation, the LC-Quad dataset is utilized, with a 70:30 split for training and testing. It contains 30,000 questions, candidate queries, final SPARQL queries, and responses, making it the largest data set accessible for complex question answering (LCQuAD) across knowledge graphs. It works with Wikidata and DBpedia's 2018 knowledge graphs.

outperform all existing systems and strong LSTM baselines: predicting the semantic relatedness of two sentences and sentiment categorization. The LSTM designs discussed in the preceding section have the constraint of only allowing strictly sequential information propagation. Here, the Child-Sum TreeLSTM and the N-ary Tree-LSTM are two natural extensions of the fundamental LSTM architecture. Both

A dataset consisting of 50 natural language-based questions was generated to query the custom dataset created. The questions cover the 3 types of queries used to train the model: select, count, and boolean. It contains 30 list questions, 10 count questions, and 10 boolean questions. The proposed solution was used to construct SPARQL queries from natural language-based questions. The metrics of the results generated are tabulated in Figures 10 and 11

Figure 10 shows the accuracy for the three different types of queries predicted on the custom dataset and Figure 11 shows the average time taken to generate a query of that type Figure 17, 18, and 19 show the result of a sample SELECT, COUNT and BOOLEAN type query along with the NLP Questions respectively. Figure 14 displays the custom dataset created and used for prediction.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1


```
?u_1 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_0 <http://dbpedia.org/ontology/creator> ?u_1 .?u_0 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
?u_0 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_0 <http://dbpedia.org/ontology/creator> ?u_1 .?u_1 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
?u_1 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_0 <http://dbpedia.org/ontology/creator> ?u_1 .?u_1 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
?u_0 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_0 <http://dbpedia.org/ontology/creator> ?u_1 .?u_0 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
?u_1 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_1 <http://dbpedia.org/ontology/creator> ?u_0 .?u_0 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
?u_0 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_1 <http://dbpedia.org/ontology/creator> ?u_0 .?u_1 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
?u_1 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_1 <http://dbpedia.org/ontology/creator> ?u_0 .?u_1 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
?u_0 <http://dbpedia.org/ontology/creator> <http://dbpedia.org/resource/Bill_Finger> .?u_1 <http://dbpedia.org/ontology/creator> ?u_0 .?u_0 <http://www.w3.org/1999/02/22-rd
f-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>
```

Figure 10. Candidate Queries for a Sample Question

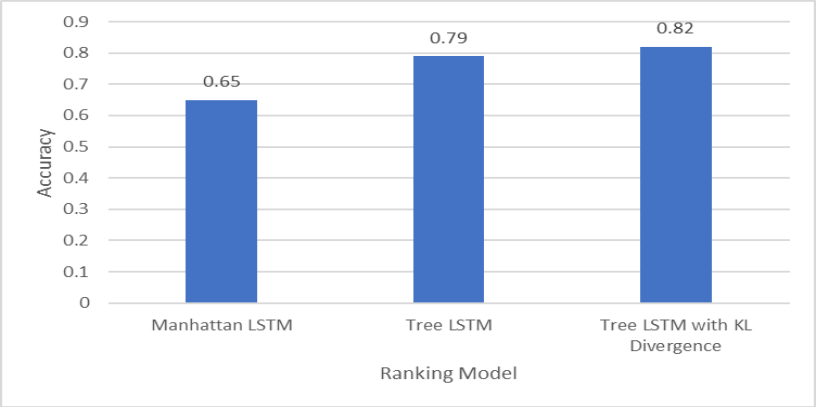

Figure 11. A bar chart comparing the accuracies of different ranking models.

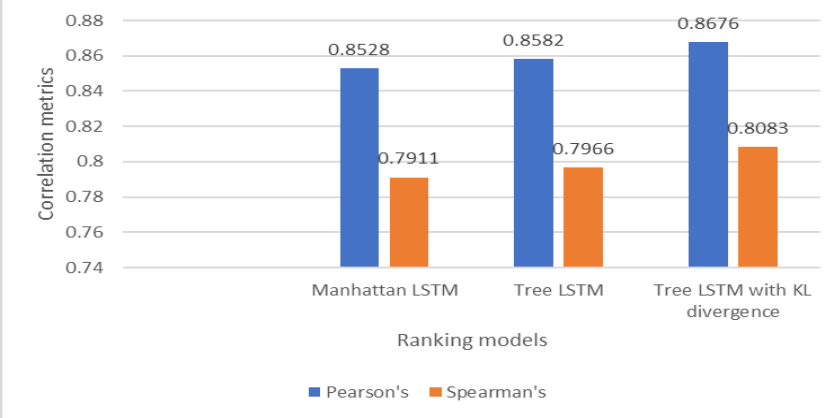

Figure 12. A bar chart comparing **Pearson Correlation and Spearman's correlation** metric of different ranking models.

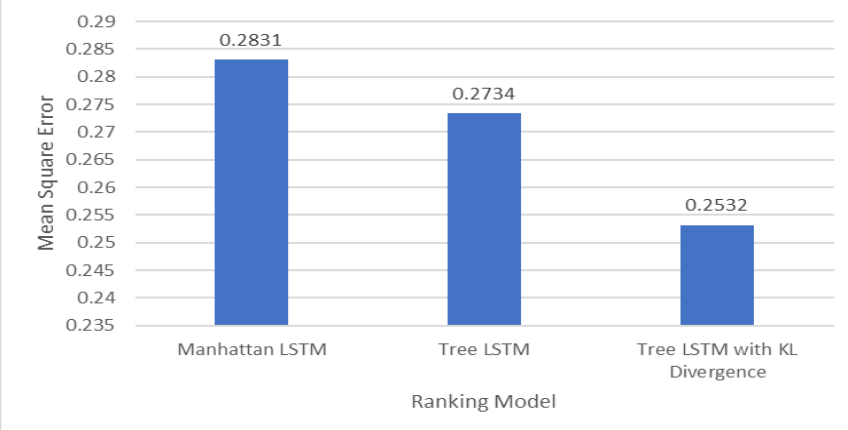

Figure 13. A bar chart comparing **Mean square error** of different ranking models.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1

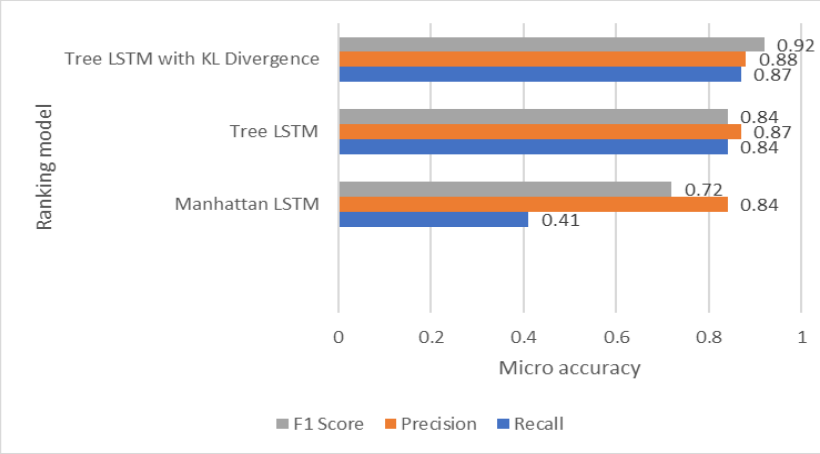

Figure 14. A bar chart comparing **F1 Score, Precision, and Recall measures** of different ranking models.

Figure 15. Prediction Analysis with accuracy as the parameter

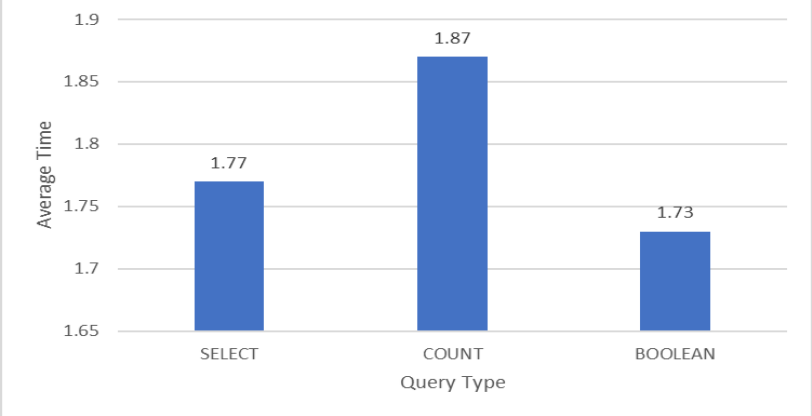

Figure 16. Prediction Analysis with Average Time as the parameter

```
Question:
what is the age of patient caleb ?
Query where clause:
['<http://example.org/patient> <http://example.org/has> ?u_0', '<http://example.org/Caleb> <http://example.org/age> ?u_1', '?u_0 <http://example.org/age> ?u_1']
Sparql Query:
 SELECT DISTINCT ?u_1 WHERE { <http://example.org/patient> <http://example.org/has> ?u_0 . <http://example.org/Caleb> <http://example.org/age> ?u_1 . ?u_0 <http://example.o
rg/age> ?u_1 }
Answer:
{'head': {'link': [], 'vars': 'u_1'}, 'results': {'distinct': False, 'ordered': False, 'bindings': [{'u_1': {'type': 'typed-literal', 'datatype': 'http://www.w3.org/2001/XM
LSchema#integer', 'value': '63'}}]}}
```

Figure 17. Sample SELECT Type Question and query result

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 December 2023 doi:10.20944/preprints202312.0228.v1


```
Question:
How many patients are there in the hospital?
Query where clause:
['<http://example.org/patient> <http://example.org/has> ?u_0', '?u_1 <http://example.org/has> ?u_0']
Sparql Query:
 SELECT (count(?u_1) AS ?count) WHERE { <http://example.org/patient> <http://example.org/has> ?u_0 . ?u_1 <http://example.org/has> ?u_0 }
Answer:
{'head': {'link': [], 'vars': 'count'}, 'results': {'distinct': False, 'ordered': False, 'bindings': [{'count': {'type': 'typed-literal', 'datatype': 'http://www.w3.org/200
1/XMLSchema#integer', 'value': '828'}}]}}
```

Figure 18. Sample COUNT Type Question and query result

```
Question:
Does Daniel have symptoms of dizziness?
Query where clause:
['<http://example.org/Daniel> <http://example.org/dizziness> ?u_0', '?u_1 <http://example.org/dizziness> ?u_0', '<http://example.org/Symptoms> <http://example.org/has> ?u_1
']
Sparql Query:
 ASK WHERE { <http://example.org/Daniel> <http://example.org/dizziness> ?u_0 . ?u_1 <http://example.org/dizziness> ?u_0 . <http://example.org/Symptoms> <http://example.org/
has> ?u_1 }
Answer:
{'head': {'link': [], 'vars': 'count'}, 'results': {'distinct': False, 'ordered': False, 'bindings': [{'count': {'type': 'typed-literal', 'datatype': 'http://www.w3.org/200
1/XMLSchema#integer', 'value': '1'}}]}}
```

Figure 19. Sample BOOLEAN Type Question and query result

	A	B	C	D	E	F
	Question	Query	Answer	Time taken		
	SELECT TYPE					
	What is the age of patient Caleb ?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	51.53		
	List all patient names in the hospital ?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	41.36		
	What does the cvd results of Tracey for coronary angiogram say?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	60.29		
	What does Bonnie's cvd results show for the presence of cholesterol ?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	66.59		
	What is the max heart rate of Eileen in cvd data?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	63.73		
	What is the sex of patient Emily?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	65.67		
	Are there any detection of blocks in Laurie found in the cvd data?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	63.73		
	What is the max heart rate Yvonne in cvd?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	63.89		
	What is the data recorded for chest pain in Debra?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	67.28		
	What is the chol level of patient Anthony	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	53.73		
	What is the max heart rate of Edward?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	65.43		
	What is the value of old peak symptom of Luke?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	65.45		
	What is the Juan's fbs value recorded ?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	50.16		
	What is the value of restecg patient Evan ?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	66.44		
	What is the age of patient Molly?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	74.51		
	What is the presence of cholestrol of Sherri in cvd?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	63.79		
	What does April have in the records for caa value?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	48.76		
	What is the value of blood vomitting symptom of Mercedes?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	63.74		
	What is the max heart rate of Edward?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	75.6		
	What is the fatigue symptom of Mason?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	72.43		
	What is the value of presence of cholesterol patient in Tamara ?	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	74.15		
	What is exng result of patient George	SELECT DISTINCT ?u_1 WHERE { <http://exar	{'head': {'link': [], 'vars': 'u_1'}, 'results':	81.76		

Figure 20. Screenshot of the custom dataset generated

6. Conclusion

Ontology is a branch of philosophy that studies the nature of life. It attempts to identify and describe the basic categories that exist, such as objects, properties, events, and relationships, and to determine the relationships between these categories. In recent years, ontology has also become a hot topic in computer science and information technology, where it is used to create language and data structures that can be shared and reused in systems and directories. Integration of NLP and ontologies can be beneficial in many ways, such as increasing the accuracy of data retrieval. Ontologies provide data structures that can help NLP systems access higher-level context and meaning from text. By integrating NLP with ontologies, the accuracy of data retrieval can be improved because NLP techniques can use ontology data to extract and interpret relevant information from text. Augmented Natural Language Understanding Ontology provides a framework that helps NLP processes understand the meaning of words and phrases in context. Do the information wisely; NLP build smart apps and answer questions, extract data, make recommendations etc. It integrates with ontologies that can be used to perform complex tasks such as NLP systems, can use ontology knowledge to provide better and relevant results, thus improving the customer experience. Enables interoperability; Ontologies facilitate the integration of different systems and applications by providing a common language and structure for data representation. By

integrating NLP with ontologies, the same ontology data can be shared and reused in different applications, leading to more efficient use of resources. Overall, integrating NLP with ontologies leads to a more accurate, efficient and intelligent language that supports a wide variety of applications and use cases. Additional work will include testing new encoder designs and pre-production graphics to improve product design.

In this project, we use the integration method to create NLP-based integration of different data sets and perform queries within it. It provides better performance and results. The system also creates diversity of information as it comes from different sources and regional interests. It has been shown to provide greater accuracy as it reduces errors and inconsistencies by comparing and cross-checking evidence.

References

1. Patel A. S. et al., An NLP-guided ontology development and refinement approach to represent and query visual information, *Expert Systems with Applications*, **2023**, 213, 118998, doi: 10.1016/j.eswa.2022.118998.
2. ASKAR, M.; ALGERGAWY, A.; SOLIMAN, T. H. A.; Birgitta K. O.; Adel, A. S., Ontology Based Natural Language Queries Transformation into SPARQL Queries, *Baltic J. Modern Computing*, **2020**, 8, 719–731.
3. Saha, D.; Floratou, A.; Sankaranarayanan, K.; Minhas, U. F.; Mittal, A. R.; Ozcan, F., ATHENA: an ontology-driven system for natural language querying over relational data stores, *Proceedings of the VLDB Endowment*, **2016**, 9, 1209-1220.
4. Coelho, M. de C.; Júlio Cesar dos Reis, Learning to build SPARQL queries from natural language questions, Graduation thesis, Instituto de Computa¸cao, Universidade Estadual de Campinas, Campinas, Brazil, Dec 2019.
5. Trivedi, P.; Gaurav, M.; Mohnish, D.; Jens, L., LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs, In *International Workshop on the Semantic Web (2017)*, Springer, Cham, Switzerland, 2016, 300–316, DOI:10.1007/978-3-319-68204-4_22.
6. Liang, S.; Stockinger, K.; De Farias, T. M.; Anisimova, M.; Gil, M., Querying knowledge graphs in natural language, *Journal of Big Data*, **2021**, 8, doi: 10.1186/s40537-020- 00383-w.
7. Shoaib, M.; Basharat, A., 4th Generation Flexible Query Language for Ontology Querying, In *2011 10th IEEE/ACIS International Conference on Computer and Information Science*, IEEE, 2011, doi: 10.1109/icis.2011.60
8. Tai K.; Socher, R.; Manning, C., Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, *CoRR*, **2015**, doi:https://doi.org/10.48550/arXiv.1503.00075
9. Kusuma, S. F.; Siahaan, D. O.; Fatichah, C., Automatic Question Generation In Education Domain Based OnOntology, In *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, 2020, doi: 10.1109/cenim51130.20 20.9297991.
10. Arguello, M.; Des, J.; Fernandez-Prieto, M. J.; Perez, R.; Lekkas, S., An Ontology-Based Approach to Natural Language Generation from Coded Data in Electronic Health Records, In *2011 UKSim 5th European Symposium on ComputerModeling and Simulation*, Madrid, Spain, 2011, doi: 10.1109/ems.2011.47.
11. Ting, M.; Kadir, R. A.; Azman, A.; Sembok, T. M. T.; Dato Ahmad, F.; Sharef, N. M., Query refinement for ontology information extraction, In *2016 Third International Conference on Information Retrieval and Knowledge Management (CAMP)*, Malacca, Malaysia, 2016, doi: 10.1109/infrkm.2016.7806340.
12. Yin, X.; Gromann, D.; Rudolph, S., Neural Machine Translating from Natural Language to SPARQL, *CoRR*, **2019**, abs/1906.09302, https://doi.org/10.48550/arXiv.1906.09302.
13. Bordes, A.; Chopra, S.; Weston, J., Question Answering with Subgraph Embeddings, *CoRR*, **2014**, arXiv:1406.3676.
14. Perevalov, A.; Diefenbach, D., Usbeck, R.; Both, A. QALD-9-plus: A Multilingual Dataset for Question Answering over DBpedia and Wikidata Translated by Native Speakers, In *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, IEEE, 2022, 229-234.
15. Dubey, M.; Banerjee, D.; Abdelkawi, A.; Lehmann, J., LCQuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia, In *The Semantic Web – ISWC 2019: 18th International Semantic Web Conference*, Auckland, New Zealand, 2019, 69–78.
16. Vrandečić, D.; Krötzsch, M., Wikidata: A Free Collaborative Knowledgebase, *Communications of the ACM*, **2014**, 57, 78–85.

 doi:10.20944/preprints202312.0228.v1

17. Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z., DBpedia: A Nucleus for a Web of Open Data, In *The Semantic Web*, Springer, 2007, 722–735.
18. Cheatham, M.; Hitzler, P., Conference v2.0: An Uncertain Version of the OAEI Conference Benchmark. In *International Semantic Web Conference (2)*, Springer, 2014, 33-48.
19. Staudemeyer, R. C.; Morris, E. R., Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks, *CoRR*, **2019**, abs/1909.09586.
20. Tai, K. S.; Socher, R.; Manning, C. D., Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, *CoRR*, **2015**, abs/1503.00075.