Natural language query formalization to SPARQL for querying knowledge bases using Rasa

Divyansh Shankar Mishra¹ · Abhinav Agarwal¹ · B. P. Swathi¹ · K C. Akshay¹

Received: 16 April 2021 / Accepted: 6 November 2021 / Published online: 11 December 2021
© The Author(s) 2021

Abstract

The idea of data to be semantically linked and the subsequent usage of this linked data with modern computer applications has been one of the most important aspects of Web 3.0. However, the actualization of this aspect has been challenging due to the difficulties associated with building knowledge bases and using formal languages to query them. In this regard, SPARQL, a recursive acronym for standard query language and protocol for Linked Open Data and Resource Description Framework databases, is a most popular formal querying language. Nonetheless, writing SPARQL queries is known to be difficult, even for experts. Natural language query formalization, which involves semantically parsing natural language queries to their formal language equivalents, has been an essential step in overcoming this steep learning curve. Recent work in the field has seen the usage of artificial intelligence (AI) techniques for language modelling with adequate accuracy. This paper discusses a design for creating a closed domain ontology, which is then used by an AI-powered chat-bot that incorporates natural language query formalization for querying linked data using Rasa for entity extraction after intent recognition. A precision–recall analysis is performed using in-built Rasa tools in conjunction with our own testing parameters, and it is found that our system achieves a precision of 0.78, recall of 0.79 and F1-score of 0.79, which are better than the current state of the art.

Keywords Rasa · NLU · Natural language query formalization · SPARQL · Ontology

1 Introduction

The field of Natural Language Processing (NLP), in the last decade, has seen a tremendous amount of work being done toward bridging the human–computer divide. The idea behind this theoretically advanced concept is to take natural human language and convert it to machine-understandable form by breaking the chunks of text into paragraphs, sentences, and words and processing them, with or without an intermediate structure, for different ends. While once a tedious task, various standard parts of speech (PoS) taggers, such as the Stanford POS tagger or Brill Tagger, have now been developed to achieve the same automatically and efficiently. NLP has also emerged as one of the most important

sub-domains in the field of artificial intelligence with recurrent neural networks (RNNs) [18,20] and long short-term memory (LSTM) [6] networks now replacing more traditional methods such as logical regression and support vector machine (SVM). Thus, there is a shift in the field toward automating the language processing and modelling pipeline for easy and scalable implementation.

With the advent of Web 3.0, there has also been a marked shift from traditional databases to larger, more complex semantically linked knowledge bases. Certain massive scale knowledge bases such as the Google knowledge vault [8], DBpedia [25], Freebase [3] and Yet Another Great Ontology (YAGO) [12] are the prime examples of the same. As such, devising a user-friendly approach for querying them to get accurate results has become an important avenue of research. While formal querying languages, like SPARQL, exist, they usually have a steep learning curve attached to them, in parts, owing to the size and complexity of the data they query and thus can be difficult for casual users [23]. In this regard, a systematic approach that allows users to query said large-scale knowledge bases without having in-depth domain knowledge is required.

✉ B. P. Swathi
swathi.bp@manipal.edu

K C. Akshay
akshay.kc@manipal.edu

¹ Department of Information and Communication Technology,
Manipal Institute of Technology, Manipal Academy of
Higher Education, Manipal, Karnataka, India

While keyword-based search engines such as that by Wilson et al. [32] have been shown to work well with unstructured large-scale textual data, they have failed to be adequately expressive to handle structured data, much less structured data of the knowledge-base scale [7]. Traditional form-based systems like [16], which provide a structure for users to elaborate their queries, have also been suggested; however, while they are comparatively more accurate, they have failed to be as user friendly. Beyond these traditional approaches, natural language interfaces (NLI) have been suggested as a user-friendly way to query structured data of this scale, and past studies [10,11] have shown this approach to have a better user experience than traditional approaches. However, an NLI needs a robust mechanism that takes user queries in natural language and formalizes them to fetch desired results.

The idea of natural language queries being converted to formal querying languages is implemented through different natural language query formalization (NLQF) [4] pipelines such as AskNow [9] by Dubey et al. that uses an intermediary canonical syntactic form called Normalized Query Architecture (NQS) or PAROT [28] (Recursive acronym) by Peter Ochieng, which uses a set of dependency-based heuristics to convert the natural language user queries to user's triples which are then processed into ontology triples. Most NLQF pipelines, however, suffer from issues about scalability, accuracy, and efficiency [13,31]. This can be attributed to challenges such as natural language being generally ambiguous and concealed in semantics and context, a general lack of widely used query-type standard, lexical mismatch of query tokens and intent mismatch due to missed contextual information. Furthermore, real intent/desire (or answer type) identification in case of differently phrased queries for the same effect remains a substantial challenge.

Rasa, an open-source machine learning (ML)-based framework to automate text and voice-based assistants, has been suggested to build intelligent chat-bot systems and has shown to have a remarkable capacity to identify intents, pick entities, and handle context [21]. It comprises multiple natural language processing and machine learning libraries that are used to make scalable conversational AIs. Rasa can pick entities and place them in user-defined slots to be accessed in the back-end later; it can also use these slots to understand the context in which the user has raised a query. Furthermore, Rasa can identify user intent/desire with a large set of paraphrased queries with high accuracy. It can be integrated into a plethora of messaging services, and individual messages from recorded conversations can be fetched and correctly annotated for easy debugging. It for these reasons that Rasa becomes an integral part of our NLQF pipeline.

The main aim and scope of this paper includes:

- Design and creation of our own ontology for a hostel management system using Protege tool.
- Design and implementation of a NLQF pipeline employing a chat-bot made with Rasa's help for intent identification and entity picking.

The proposed ontology has been designed, keeping in view the current hostel system at our university. The questions or generalized user intents have been chosen from the Frequently Asked Questions (FAQ) section of the college hostel website. The ontology made is queried by users using natural language in English. Each intent has a corresponding skeleton SPARQL query plugged with picked entities such as objects, events or concepts and run after identifying that specific user intent or query desire. A classical precision–recall analysis is performed using in-built Rasa tools and some external parameters to measure the system's accuracy, taking in consideration how accurately the system can pick intents and thus build queries and how accurately the system can return desired results.

This paper is organized as follows: Section 2 is briefly touched upon related work in natural language query processing, query formalization and recent advancements made in the field, Sect. 3 and following subsections discuss our methodology for the pipeline in detail, vis-à-vis, the ontology design, the Rasa framework and the python scripts to query the knowledge base. Section 4 is used to discuss the experimental setup, the analysis conducted and the results we have achieved with this pipeline and finally Sect. 5 presents our conclusions along with a brief overview of possible future work in the field.

2 Literature review

This section is used to discuss the past work, the recent advancements, and the current state of the art in the field of natural language query formalization for both structured and unstructured data. An account of approaches used in these works along with associated drawbacks are as follows:

With "Natural Language Query Processing for SPARQL generation—a Prototype System for Systematized Nomenclature of Medicine Clinical Terms (SNOMEDCT)" [24], Jin-Dong Kim, and Kevin B. Cohen have discussed the Linked Open Data Question Answering (LODQA) system, which is used to generate SPARQL queries from natural languages queries to search open data. They have made a prototype version of LODAQ, which works on SNOMED CT. Their methodology includes pseudo-SPARQL generation from natural language queries using Enju and then converting this pseudo SPARQL to SPARQL query. However, with this implementation, Enju does not produce correct

analysis for queries of the type “what is x for y” and “y for which x.”

In their 2013 paper [14], Ivan Habernal and Miloslav Konopík discuss a natural language-based method to search the semantic web. They have identified the formal querying language, SPARQL, to be best suited to get results from a semantically connected knowledge base. To this end, they have formalized natural language queries in both Czech and English through pre-processing, semantic analysis, and semantic identification. The SPARQL queries are then evaluated on two domains: “accommodation options” and “public transportation” in Czech and English, respectively. However, their approach faces issues with ontology reasoners, especially with transitive properties, mainly due to the large size of their dataset.

Baudiš et al. [2], briefly survey contemporary techniques for question answering systems and present the YodaQA system, a baseline pipeline with an open-source structure. YodaQA is shown to have good performance for the question-answering task and uses a machine learning model in conjunction with knowledge-base paradigms. The authors have also proposed to build a new dataset for reference QA testing and an extended version of the TREC corpus. The approach, however, is still listed as a work in progress.

In their 2016 paper [29], Mahaboob Hussain and Prathyusha Kanakam discuss their methodology for reforming natural language queries into SPARQL queries. They have made a three-module pipeline consisting of a graphicator that generates a pseudo graph pattern, a termfinder that looks for matching terms in the knowledge base, and generates an anchored pseudo graph pattern and a graph-finder finds similar triples in the knowledge base as the anchored graph. The model is robust and however may suffer from incorrect anchoring.

In this paper [9], the authors propose a framework called AskNow, where users can raise questions in English on a target RDF knowledge base (such as DBpedia), which is first generalized to an intermediate canonical syntactic form called normalized query structure (NQS) and then translated to SPARQL queries. NQS helps identify desires (or expected output information) and input information provided by the user and establish their mutual relationship. At the same time, it is adaptive enough to work around query paraphrasing. The authors empirically show that using benchmark datasets, NQS is robust in terms of syntactic variation and highly accurate in identifying the query desire. This approach performs well with large-scale knowledge bases; however, conversion to an intermediate form and its processing can become a bottleneck. Another important suggestion in this paper is the use of a predefined query-type set, which we have modified in our approach.

In “Learning a Neural Semantic Parser from User Feedback” [19], Iyer, Konstas, Cheung, Krishnamurthy & Zettle-

moyer have made a bidirectional LSTM for natural language formalization to Structured Query Language (SQL). They have used an encoder–decoder-based approach wherein the decoder predicts the SQL token embeddings’ conditional probability. They have also used human feedback in conjunction to better their model. This approach provides for better training and fault correction. Their approach, however, cannot be used to query large-scale knowledge bases.

Hao et al., in their 2017 paper [15], have also applied a bidirectional long short-term memory(LSTM) network to convert natural language queries to SPARQL queries. The bidirectional LSTM is used to pick the word’s context about words immediately following it and immediately preceding it. The top keyword in the Natural Language (NL) query is extracted and used to pick a suitable answer from the knowledge base. These words are then linked to the correct answer tokens by learning their relatedness. This approach works well with keyword type queries, however, will perform inadequately with scalar queries.

Sébastien Ferré, in this 2018 paper [13], proposes SPARKLIS (recursive acronym), a tool that helps users to build queries for SPARQL endpoints. The tool works by guiding the user through interactive questions and building both simple and complex questions and fetch their answers. The users do not need to have any prior knowledge of SPARQL vocabulary or the ontology schema. Furthermore, no endpoint specific configuration is required by the tool. Two languages, English and French, can be used with the tool with adequate accuracy. The tool is available as a web application called SPARKLIS, and the author states that they have performed thousands of queries from hundreds of users over multiple endpoints. However, it has been noted that for scalar queries, queries with negation and compound sentences, the accuracy takes a sharp hit.

With this 2017 paper, Chen et al. [5] offer solutions to open domain questions answered using Wikipedia as a single source of knowledge, considering the answer to any factual question being the text in the Wikipedia article. This task of machine-scale reading connects the problems of searching for documents (finding relevant articles) to machine comprehension of the text (determining the response range of these records). This approach combines adaptation, bigram hashing, and a Term Frequency - Inverse Document Frequency (TF-IDF)-compatible search component with a multi-layered repetitive neural network model trained to identify responses in Wikipedia paragraphs. The result of this paper shows that machine reading at scale (MRS) is a major challenge for researchers to give attention to. Machine comprehension systems cannot solve the general task alone.

With “SQLizer: query synthesis from natural language” [34], Navid Yaghmazadeh et al. have proposed a method that combines contemporary semantic parsing techniques with type-directed program synthesis and automated pro-

gram repair to develop an end-to-end system to synthesize SQL queries from NL queries. The model works without any requirement of underlying schema and performs consistently on a variety of databases. Probabilistic type inhabitation and automated sketch repair are alternated for iterative refinement. This proposed method outperforms NALIR [26] but cannot effectively be scaled over semantically linked data.

Yuri Murayam et al. in this 2018 paper [27] have proposed a methodology to generate SPARQL queries, by deploying a six-stage pipeline, taking a natural language query as input and returning a complete a SPARQL query. This methodology can be used to harness information from the growing knowledge of the web. The researchers have then deployed their methodology to a scenario where a customer orders food at a cafe. They have recorded good results for fetching information from an ontology, given a natural language query. However, the performance of this approach on large-scale knowledge bases will not be adequate.

The authors in this 2018 paper [1] talk about an overall evolution of data in semantic form, and how this evolution has led to the development of natural language answering systems that automatically translate a question into SPARQL based on RDF knowledge graphs. Hybrid question-answering, the task of answering questions by combining both structured (RDF) and unstructured sources of knowledge (text), is discussed along with challenges pertaining to such hybrid systems. This paper deals with hybrid answers to natural language questions. The authors present HAWK_R (recursive acronym), a question-answering system that enhances the open-source system called HAWK. The authors identify its limitations and suggest improvements using heuristic methods based on RDF and text search. Their results show a clear improvement over contemporary methods; however, recent AI-based advancements for querying structured data have reduced the impetus given to unstructured textual data.

In this 2018 paper [17], the authors talk about natural language Q&A systems on RDF data. Although several studies have solved a small number of aggregate questions, they have many limitations (e.g. interactive information, controlled questions, or query templates). To date, there is no natural language search method that can process general overall queries on RDF data. Therefore, the authors proposed a structure called Natural Language Aggregate Query (NLAUQ). Firstly, the authors discuss a novel algorithm to automatically understand a user's query's intent, which consists primarily of semantic relationships and aggregates. Secondly, to create a better bridge between query intent and RDF data, the authors propose an extended paragraph dictionary, ED to get more candidate mappings for semantic relationships and introduce predicate-type adjunct set PTs and words to filter inappropriate candidate mapping combinations. Thirdly, the authors design an appropriate translation plan for each

overall category that can effectively differentiate whether an item is numerical or not that greatly affects the overall result. Towards the end, the authors conducted extensive experiments on Real Datasets (KALD Benchmark and DBPedia), and the experimental results show that their solution is effective. While candidate mapping is a good approach, there has been much work done with automatic intent classification, which can improve this model substantially.

In their 2018 paper, Xu et al. [33] have discussed problems associated with the classic sequence-to-sequence-style model, which necessitates the serialization of SQL Queries. Unlike contemporary approaches that use reinforcement learning models to train the decoder to generate equivalent serializations of a query, they have proposed SQLNet, which utilizes dependency graphs such that each prediction depends on only the previous predictions. Furthermore, the employment of other novel aspects such as sequence-to-set-model and column attention mechanism has improved their results over the state of the art by up to 13% on WikiSQL. However, with semantically linked data, equivalent serializations may fail to provide accurate results in a variety of use cases, and thus, this approach cannot be scaled to knowledge bases.

This 2019 paper [22] talks about websites based on Community based Question Answering (CQA) that allow users to post their questions and have their questions answered by other users. The answers on these CQA websites can range from specific questions related to a particular field of interest to the users to other kinds of more generalized questions. Creating automated CQA websites is of great importance for the research into natural language processing. One of the tasks in developing automated CQA websites is to find similar questions as the user's question. This article proposes a new method for finding questions related to a user's question with deep LSTM neural networks. Experimental results show that the proposed algorithm has high accuracy for finding questions in CQA social networks. A drawback, however, is that in most cases answers are also fetched based on answers of similar questions, which are community generated, and thus, accuracy can be low.

In his 2020 paper [28], Peter Ochieng has discussed a dependency-based framework for natural language query formalization called PAROT that can handle queries with scalar values, negations, numbered list, and compound sentences. He has designed a lexicon that tackles contextual ambiguity by fully representing the knowledge base's vocabulary and that tags each adjective with its positive and negative scalars. The entire system has been coded from scratch, and it does not make use of any existing framework. Ontology triples are generated from user triples through processing using this lexicon, which are finally used to generate SPARQL queries. Although a robust system, PAROT is not scalable to large knowledge bases due to its slow lexicon generation. Further-

more, it performs inadequately with aggregate queries and queries, beginning with the word "when".

In his 2020 paper [21] and neural network, Anran Jiao has performed a comparative study between two methods to develop conversational AI: using Rasa-NLU and using a RNN. The two models are trained on a set of questions related to stocks and finance and then validated and compared for intent recognition and entity extraction. They have found that Rasa-NLU outperforms the RNN in accuracy for a single experiment; however, the RNN works better with entity extraction for segmented words. Some of the drawbacks of this work include its inability to support larger and more complex sentences and that the method is only designed for academic use, not commercial use.

Thus, we conclude that the methods mentioned above and approaches while being adequate in some contexts, fall short in some. Issues pertaining to no scalability, robustness and scope of generalization persist in some shape or form in most of these approaches. Moreover, a lot of these approaches employ complex pipelines that are juxtaposed with the problem of simplifying querying knowledge bases that they are meant to solve, thus leaving scope for optimization. With this paper, we have tried to learn from and perform a precision–recall analysis using in-built Rasa tools in conjunction with our own testing parameter that are discussed in the following section.

3 Methodology

Our method is an end-to-end pipeline powered by Rasa at its core. The supporting querying scripts are written in Python. The pipeline starts with training the Rasa NLU model with the training sets of all possible intents or query desires. After the model is trained, the bot becomes ready to accept natural language user queries.

User queries are received, their intent/desire is identified, and entities are picked after that. After successful intent identification, a corresponding action is invoked where the pre-designed skeleton query is plugged with picked entities. The query so generated is used to fetch results from the hostel system knowledge base, packaged in a conversational aspect, and returned to the user through the same corresponding action. The user can choose to ask questions related to the last question/answer (context) or end the conversation with the bot returning it to its inactive state.

A diagrammatic representation of this pipeline is shown in Fig. 1. The following subsections discuss the knowledge base and the individual components of this pipeline in detail.

Fig. 1 A complete pipeline overview

3.1 Ontology creation

Ontologies have become powerful tools in designing knowledge bases with complex semantic relations between different classes. With the general shift towards linked data, ontologies' use to model semantic relations between classes has become widespread, and their application in fields varying from technologically enhanced learning (TEL) to ML exemplifies the same.

For this approach, Protégé, an open-source ontology design, and editing framework, has been used to build the hostel system ontology using the Web Ontology Language(OWL)/RDF format from scratch. With its classes, properties, and individuals, the ontology mimics the actual hostel rules and regulations at our University.

The ontology consists of two base classes: block and room corresponding to the hostel blocks and types of rooms in them. Each hostel has properties corresponding to the resident's gender, i.e. if it is for male students or females, the hostel's total housing capacity, if it is for juniors or seniors if it has a mess/dining area inside it and finally its distance from the library. The room class has properties corresponding to the occupancy of the room (single, double, triple), if it is air-conditioned if it has an attached washroom, the minimum grade point assessment (GPA) requirement for the room, and the room's yearly fee. Each hostel has at least one type of room, and some have all types of room. The total number of individuals for the block class is 20 for 20 hostel

Fig. 2 The hostel system ontology

blocks, while the number of individuals for the room class is 7, corresponding to the seven types of rooms available on campus.

The ontology so designed is fairly complex and is an accurate model of the college hostel system to the best of our knowledge. The diagrammatic representation of the hostel system ontology is shown in Fig. 2. The Owlready2 python library is used to import the ontology in the actions.py python script in RDF format in this approach.

3.2 Rasa pipeline

The Rasa pipeline consists of components for pre-processing the training data set, intent classification, entity extraction, and response selection. The pipeline is relatively flexible where individual customized components can be added or existing models can be modified depending upon the user's requirements. Broadly, the pipeline consists of three main parts vis-à-vis tokenization, featurization, and entity recognition/intent classification/response selection. The configurations are stored in the config.yml file.

For this approach, WhiteSpaceTokenizer has been used for the tokenization step. It works, as the name suggests, by tokenization of every whitespace-separated character sequence. The choice is largely due to the variety of possible entities for each intent; however, ConveRTTokenizer can be used interchangeably without much effect to accuracy. For the featurization step, a set of three sparse featurizers, RegexFeaturizer, LexicalSyntacticFeaturizer, and CountVectorsFeaturizer, have been used instead of the available pre-trained featurizers like Spacy or ConveRTFeaturizer due to the variety of intents and substantial training data. RegexFeaturizer creates a list of regular expression defined in the training format for which a feature is then set marking whether that

Fig. 3 The Rasa pipeline

expression was discovered in the user input or not, the LexicalSyntacticFeaturizer works using a sliding window that slides over each token and creates features for entity extraction, lastly, the CountVectorFeaturizer utilizes the sklearn's CountVectorizer to create a bag-of-words representation of user messages. Each of these featurizers can be configured to individual requirements; however, we have used them in their default configurations in our approach. Finally, for the intent classification/entity recognition/response selection step, we use the dual intent and entity transformer (DIET) classifier to provide state-of-the-art accuracy and operate on picked features from the featurization step. The DIET classifier is discussed in the following subsections.

A diagrammatic representation of the Rasa pipeline is shown in Fig. 3, and the corresponding configuration file is depicted in Fig. 4. The following subsections discuss the details of the components used to train the bot and operate it.

3.2.1 Intents

In any NLQF pipeline, identifying query intent/query desire, albeit challenging, is a critical step on which the quality

```

language: en
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
- name: EntitySynonymMapper
- name: ResponseSelector
  epochs: 100

# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies:
- name: MemoizationPolicy
- name: TEDPolicy
  max_history: 5
  epochs: 100
- name: MappingPolicy

```

Fig. 4 Rasa configuration file

of the returned answer depends. Major challenges associated with accurate intent identification are: (1) Paraphrased queries such as "How many students live in Block 10?" and "Total strength of 10th Block"; (2) The context in which a word is used such as "How close is this hostel from the library?" and "What time does the library close"; (3) Usage of general synonyms such as "Size of Air Conditioned (AC) room with attached washroom" and "Dimensions of AC room with attached washroom"; (4) Accuracy in types of answers returned such as "Is this hostel for girls" needs a Boolean answer while "List all hostel for girls" requires a list of hostel names. Rasa, with enough data, can be trained to work around most of these issues and deliver accurate results. This is discussed in depth in Sect. 4.

An intent in Rasa is defined by a collection of training sentence set, each of which serve the same purpose but might show great variation in structure, for example: "Can you tell me what is the yearly fee of an AC room with attached washroom?" and "AC attached washroom room amount" sentences greatly varied in semantics, but the goal of both is to inquire about the fee of an air-conditioned room with an attached washroom. Rasa utilizes the Dual Intent and Entity Transformer (DIET) classifier, a multi-task architecture for intent classification and entity recognition, that uses a shared transformer for the two tasks. The key characteristic of DIET classifier is the ability to integrate with pre-trained word embeddings from language models and combine these with sparse word and character level n-gram features in a plug-and-play fashion. DIET shows high accuracy, outperforms fine-tuned bidirectional encoder representations from transformers (BERT) [30] and has been shown to be six times faster in training. Intent labels are embedded into a single semantic vector; then, dot product is used to calculate simi-

Fig. 5 Working of the DIET classifier

Table 1 Examples of Intents based on query types

Type	Intent	Query
Boolean	gender_of_hostel	"Is 10th block
	_residents_specific	hostel for boys?"
List	ask_hostel_year_list	"List all hostels for seniors"
Property	ask_hostel	"What are types
	_room_type	of room available in hostel?"
Count	room_at_gpa	"How many rooms can I apply to with a 7.6 gpa?"

larity with the target vector, thus turning intent classification into an $n+1$ classification problem. Figure 5 shows how DIET classifier is utilized.

Our dataset's intents were based around the FAQs for the college hostel system and were generally varied across several query types. All in all, we identified four query types, vis-à-vis (1) Boolean, (2) List, (3) Property and (4) Count, and segregated the FAQ section based on them. Varied sets of sentences for intents corresponding to each of these query types were recorded through actual questions posted by students, which were then paraphrased in various semantical forms to augment the training data set. A total of 30 individual intents were identified, with each having an average of 12 sentences used to train the model. These were stored in the nlu.md file. These intents ranged from "greet" and "goodbye" which are used to activate and deactivate the bot to "ask_hostel_gender_details" which is used to inquire if a hostel is for girls or boys. Examples of intents along with their query type are shown in Table 1. An example of intent in Rasa, along with its sentence set, is shown in Figs. 6 and 7.

3.2.2 Slots and entities

Identifying and picking entities is another core part of any NLQF pipeline. Picked entities are used to plug the skeleton query and return desired results. In the Rasa framework,

```

137 ## intent:ask_hostel_room_type_specific
138 - Tell me about type of hostel rooms in [15th block](hostelBlock)?
139 - What are hostel room types in [8th block](hostelBlock)
140 - Can you tell me about hostel room types in [2nd block](hostelBlock)?
141 - What are types of room available in [1st block](hostelBlock)
142 - What are some types of rooms available in [3rd block](hostelBlock)?
143 - Kinds of hostel rooms available in [2nd block](hostelBlock)?
144 - Kind of hostel rooms available in [2nd block](hostelBlock)?
145 - What kind of hostel rooms in [12th block]?
146 - What kind of hostel rooms are there in [4th](hostelBlock)?
147 - Can you tell me about kinds of hostel rooms in [3rd block](hostelBlock)
148 - Can you tell me about type of hostel rooms in [8th block](hostelBlock)
149 - What kind of rooms available in [7th Block](hostelBlock)

```

Fig. 6 Rasa Intent along with training question set

```

180 ## intent:ask_hostel_mess_list
181 - Which hostels have messes in them?
182 - Which hostels have mess inside
183 - List of hostels with mess
184 - Which all hostels have messes attached?
185 - List of hostels with mess inside
186 - Hostels with mess
187 - Can you tell me which hostels have messes
188 - Can you tell me which hostels have messes inside?
189 - Can you tell me which hostels have messes in them
190 - Can you tell which hostels have mess
191 - Tell me which hostel have mess
192 - Hostel with messes
193 - Tell hostel with mess

```

Fig. 7 Rasa intent along with training question set

we used the DIET classifier again to pick and identify entities as it provides the ability to train a large set of custom entities while also being considerably faster to train. DIET classifier works well in cases where context is necessary such as "double seater" in the context of a hostels means a room with two beds, while the same can mean a car with two seats in a car dealership context. A conditional random field (CRF) tagging layer on top of the transformer predicts a sequence of entity labels corresponding to the input sequence of tokens. The predicted entity, along with their class, can then be used in the back end. DIET also works well with multiple that appear incoherently in the training data set such as in the identification of (washroom), (occupancy) and (airConditioning) in "Price of [single](occupancy) [AC](airConditioning) room" as well as in "Annual fee of the room with [one bed](occupancy) that also has an [attached washroom] (washroom)."

Slots in Rasa are key-value stores that can be used to store the information provided by the user and store answers generated by the user query. If created with the same name as an entity, slots get automatically set to the entity value when it is picked. Slots, thus, help in back-end computation using picked entities. In our approach, we used eight types of entities and thus had eight slots as well. An example of an entity washroom is given in Table 2.

3.2.3 Stories

Stories are another major component of the Rasa library that is used to design the conversational flow for the bot by training the dialogue management modules. Stories are essentially

Table 2 Examples of Entity type-Washroom

Example	Entity
Attached toilet	Washroom
Joint bathroom	Washroom
Attached washroom	Washroom
Connected toilet	Washroom

Table 3 Two types of story: specific/short and general/long

story_short	story_long
*Greet	*Greet
-Bot responds with greet	-Bot responds with greet
*Ask specific question	*Ask general or contextual
	Squestion
-Bot acknowledges question	-Bot seeks clarification
-Bot returns answer	*Clarify question/specify
	Context
*Goodbye	-Bot acknowledges question
-End session	-Bot returns answer
	*Goodbye
	-End session

snippets of possible conversations a user might have with a bot and thus are important in training the bot responses. Stories, in Rasa, are a collection of intents, responses, and actions, which can be arranged in any order to design a conversational flow. In our approach, each story starts with the intent greet and ends with the intent goodbye, between these two intents, a user can ask as many questions as are required, and bot responds with the particular set response or performs a particular action when a corresponding intent is identified. There are two general story types roughly mapped to each intent used in our approach. However, the bot can drive conversations adequately, even with a smaller story set.

A total of 38 stories have been made for this approach with each having 3–5 intents, one or Two actions and 3–5 responses. The two general story types, along with an example for each, are shown in Table 3. An example of a story in Rasa is shown in Fig. 8.

3.2.4 Actions

Actions in Rasa are what the bot runs in response to user's question or message. The simplest action used is utterance actions, which are used to essentially send to the user, a pre-written fixed response based on the identified input. Utterance actions, in our approach, are used to give user, feedback that their query has been registered and the bot is actively working on returning the desired answers. A total of 11 utterance actions are designed


```
## hostel capacity and room
* greet
  - utter_greet
* ask_hostel_capacity
  - utter_which_hostel
* choose_hostel
  - utter_finding_hostel_details
  - action_show_hostel_capacity
* ask_hostel_room_type
  - utter_finding_hostel_details
  - action_show_hostel_room_type
* goodbye
  - utter_goodbye
```

Fig. 8 An example of a multi-intent story in Rasa

varying from the simple utter_greet: “Hello, I am here to help you with your hostel query” to the more complex utter_finding_room_details: “Give me a moment, I am looking for an [AC](airConditioning) room with [attached wash-room](washroom) in [10 th Block](hostelBlock)”. These utterance actions are pre-written in the domain.yml file, and each utterance action can have multiple possible responses which are randomly invoked once the corresponding intent is identified.

Furthermore, we employ a set of custom actions which run python scripts to generate query, use it to fetch results from the knowledge base and return these results packaged in a conversational manner to the user. These actions are also recorded in the domain.yml file and are written in action.py script. A total of 15 custom actions are used in our approach, each being linked to multiple intents and each having a skeleton query which is plugged with entities and then run.

Beyond the aforementioned, Rasa also provides a set of default actions that can be invoked for a swathe of bot related events such as session_start, deactivate_form and ask_rephrase

3.3 Querying and returning values

Each query type has multiple intents, which are then mapped to actions; these actions have skeleton queries used to query the hostel ontology. Python rdflib is used to query the knowledge base and fetch answers. These fetched answers are then packaged into a conversational aspect and returned to the user using a dispatcher. Custom actions are utilized for this procedure. The example of a skeleton query is shown in which is plugged with picked entities to generate the final SPARQL query is shown in Fig. 9.

4 Experiments and analysis

This section is used to, first, discuss the data set or the corpus used to train the model following which, the validation or

Fig. 9 Skeleton query gives final SPARQL query upon being plugged with entities

```
Your input -> hi
Questions about Manipal hostels? I'm here to help
Your input -> what is the hostel capacity
Which hostel do you want to know these details of?
Your input -> block 10
Hold on, looking up these details for block 10
The capacity of block 10 is 779
Your input -> what room type does it have
Hold on, looking up these details for block 10
Block 10 has three types of rooms:
DC
SA
SC
```

Fig. 10 A sample conversation with the bot

testing dataset is discussed, specially how it is generated, and then used. The approach is taken to evaluate different steps, and the results achieved thereafter have been discussed subsequently. The following subsections are arranged in the same order. A sample conversation with the bot is shown in Fig. 10. The conversation depicted is on the command line. However, the bot can be integrated with an array of different messaging services that support AI integration

4.1 The training set

We manually generated the training data in markdown format. The question set's main source was the FAQ section of the college hostel system, and variations on the questions were generally made through observation of actual questions listed by students. In cases where we did not have a rich student-generated question set, some variations were entirely written by us. These variations took into consideration syntactical aspects such as active and passive voice, interrogative and assertive structure, correct and incorrect verb and tense usage, and lastly, some common misspellings of frequently used words like "hostile" and "hostel" or "library" and "library". The training dataset used, thus, contained a total of roughly 400 sentences, divided across 30 intents. Out of these, roughly, around 55% are without entities, while the remaining have one or more than one entities (the majority have multiple entities). Furthermore, eight entities have been created, and 11 response types with an average of 4 possible responses have been used. To ensure that there is no possibility of overfitting, we have used testing data from an

```

"text": "With gpa of 8.19 can i get a room in 3rd-block?",
"intent": "ask_available_rooms_at_gpa_specific",
"entities": [
  {
    "end": 16,
    "entity": "gpaRequirement",
    "start": 5,
    "value": "gpa of 8.19"
  },
  {
    "end": 46,
    "entity": "hostelBlock",
    "start": 37,
    "value": "3rd-block"
  }
]

```

Fig. 11 Chatito generated sample data for validation-1

```

"text": "hello is this block for gents thank you",
"intent": "gender_of_hostel_residents",
"entities": [
  {
    "end": 29,
    "entity": "gender",
    "start": 24,
    "value": "gents"
  }
]

```

Fig. 12 Chatito generated sample data for validation-2

entirely different corpus, not related to the training data. This is explained in detail in Sect. 4.2.

4.2 Validation set generation

For validation of our model, we used Chatito, a standard dataset generation tool for validating chat-bots using Domain Specific Language (DSL), scripts to generate testing data. For this, we had to write a DSL script containing a set of intents and a collection of entities from which the sentences would be generated. We generated a total of 3000 sentences by providing 10 base sentences for each intent, using different entity sets for each variation. The sentences were generated by Chatito through a "cloud of words model". Figures 11 and 12 are samples from the JSON dataset generated using Chatito:

4.3 Training and testing procedure

This corpus, made from FAQ section question variations, consisted of over 400 sentences taking around 3 minutes to train for 100 epochs on a typical personal laptop with specs: 8 GB DDR4 RAM, 240 GB SSD, Core i5 7th gen. This is owing to the organized data set and the efficiency of the DIET classifier in Rasa, which we were able to generate models with high accuracy, quickly, without requiring substantial computational resources.

For the validation test on the model, specifically for intent classification and entity recognition, we used Rasa's inbuilt testing capabilities, for which we designed a test file conforming to the end-to-end format required for testing. To generate

this file, we also used "Rasa X," a graphical tool provided by Rasa.

The findings, discussed in the following sections, have been generated by Rasa's own in-built test utility, which can be invoked by the command `Rasa test`.

4.4 Analysis of intent recognition

The model was validated on all 30 intents. Some of the intents were very similar in a syntactical context but amounted to different query types (and thus, different actions which each then have different skeleton queries) such as "Is this hostel for men?" corresponded to a Boolean query type, while "List all hostels for men" corresponded to a list query type despite having a similar structure. Similarly, some intents were simply a "specific" variant of another more general intent such as "Hostel capacity of this block" and "Hostel capacity of 6th Block", which are again, similar in structure. As discussed in previous sections, intent classification is one of the most important aspects as well as a test for any NLQF pipeline. Consequently, our approach performs at par with state of the art in this regard. We were able to achieve near-perfect results for intent recognition and classification with an overall accuracy of 79%.

The following is a general list of intents:

1. ask_hostel_capacity*
2. choose_hostel
3. ask_hostel_room_type*
4. gender_of_hostel_residents^
5. ask_hostel_mess^
6. ask_hostel_year^
7. list_rooms_by_query
8. ask_room_gpa_requirement
9. ask_available_rooms_at_gpa*
10. ask_room_fee*
11. tell_hostel_room_type
12. ask_hostel_distance_from_library*
13. ask_library_close_time
14. ask_hostel_room_inquiry^

* indicates a specific variant also existing for this query. ^ indicates a list variant also existing for this query.

Figure 13 shows the intent prediction confidence histogram where intent sample size is varied and the corresponding number of correct and incorrect intent identifications are found.

4.5 Analysis of entity recognition

The following 8 entities have been used for different property values in our hostel ontology system.

Fig. 13 Intent recognition confidence histogram

1. hostelBlock
2. gender
3. airConditioning
4. juniors
5. gpaRequirement
6. washroom
7. occupancy
8. seniors

Juniors and seniors have been defined as different entities even though they refer to the same general idea (like gender), a class of students because of the large and varied contexts in which they have been used such as "new students", "incoming students" and "freshers" for juniors and "2nd years", "final years" and "passouts."

Our results with entity recognition were better than state of the art. The overall accuracy was found to be 79% with a recall of 0.78. The F1 score was found to be 0.78. Entity recognition was accurate for most entities but was comparatively lower for "hostelBlock" and washroom owing to a varied set of numbering conventions used such as "tenth block", "10 th block", "X block" and "10th block" for the former and an equally large, if not larger, colloquial equivalents of "attached washroom" like "joint toilet", "combined shower", "attached bath", etc.

Tables 4 and 5 show the accuracy of entity recognition for the aforementioned and other systems.

4.6 Analysis of action recognition

The following table lists all the actions deployed by us in the model. Some actions also have a list variant present.

1. show_hostel_capacity
2. show_hostel_room_type
3. tell_hostel_gender^
4. tell_hostel_mess^
5. tell_hostel_year^
6. list_hostel_by_year
7. list_rooms_by_query
8. tell_room_gpa_requirement
9. tell_rooms_at_gpa
10. tell_room_fee
11. tell_hostel_library_distance
12. ask_hostel_distance_from_library*
13. tell_library_close_time
14. which_hostel*
15. finding_hostel_details*
16. finding_hostel_list_based_on_gender*
17. finding_hostel_list_with_mess*
18. looking_up_room_details*
19. looking_distance*
20. checking_library_time*
21. which_room*

^ a list variant also exists for this action *this is actually a response which is interpreted in a similar way as an action

Rasa does not provide tools to assess the performance of invoked actions or the results they return. Thus, to check our actions' validity, and the results returned, we defined our own precision and recall metrics. In this scope, we defined a true positive as a case where the correct action is invoked after successful entity recognition, and the results returned are found to be accurate. Thus, for any case to be a true positive, we look for successful invocation and accuracy of answers returned.

In the same vein, we defined a false positive as a case where an action is invoked, but the answer returned is not correct or does not satisfy the user. This can either be due to the invocation of a wrong action and thus the generation of an improper or mismatched query or the invocation of the correct action but improper or incorrect plugging of the skeleton query.

Lastly, we define false negatives as the set of cases where either the incorrect action is invoked or no action is invoked due to intent mismatch or other ambiguous issues. We define precision as the fraction of correct answers versus all returned answers and recall as the fraction of correct answers versus all validation cases.

Table 4 Accuracy of entity recognition in proposed system and method proposed by P. Ochieng et al.

Entity	Proposed system				P. Ochieng et.al			
	Precision	Recall	F1-Score	Support	Precision	Recall	F1-score	Support
HostelBlock	0.80	0.70	0.74	279	0.65	0.60	0.62	279
AirConditioning	0.70	0.70	0.70	39	0.65	0.60	0.62	39
GpaRequirement	0.75	0.80	0.77	50	0.55	0.60	0.57	50
Seniors	0.80	0.88	0.83	8	0.59	0.65	0.61	8
Gender	0.86	0.82	0.89	31	0.60	0.62	0.60	31
Occupancy	0.71	0.75	0.72	43	0.57	0.56	0.56	43
Juniors	0.80	0.89	0.83	12	0.67	0.63	0.64	12
Washroom	0.87	0.85	0.85	56	0.60	0.59	0.59	56
Average	0.78	0.79	0.79	86	0.61	0.60	0.60	86

Table 5 Accuracy of entity recognition in system by Hao et al. and Yuri et al.

Entity	Hao et al.				Yuri et al.			
	Precision	Recall	F1-score	Support	Precision	Recall	F1-score	Support
HostelBlock	0.51	0.56	0.53	279	0.56	0.57	0.56	279
AirConditioning	0.53	0.54	0.53	39	0.66	0.70	0.67	39
GpaRequirement	0.56	0.50	0.52	50	0.64	0.66	0.64	50
Seniors	0.56	0.60	0.57	8	0.66	0.68	0.66	8
Gender	0.60	0.62	0.60	31	0.60	0.65	0.62	31
Occupancy	0.57	0.60	0.58	43	0.62	0.64	0.62	43
Juniors	0.58	0.58	0.58	12	0.59	0.58	0.58	12
Washroom	0.65	0.60	0.62	56	0.55	0.57	0.55	56
Average	0.57	0.57	0.66	86	0.61	0.63	0.61	86

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (1)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2)$$

$$\text{F1 - Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

Testing our model on the validation dataset, we were able to attain very accurate results with an average precision of 0.76% and a recall of 0.80%. The F1-score was found to be 0.78. While most of the actions were correctly invoked and returned optimal results as well, we found lower precision for the action tell_room_gpa_requirement, especially in cases where colloquial words for an attached washroom were used and as a result, an improper query was formed. We also found mismatched invocation of list_rooms_by_query and tell_rooms_at_gpa which both return suitable rooms based on certain conditions; however, this was much lower than the first issue.

Thus from the results that we have received, we can conclude that this NLQF pipeline is very accurate at intent classification and entity picking. The model also performs satisfactorily well with invoking actions, building queries and returning accurate results

5 Conclusion and future work

NLQF makes large and complex knowledge bases accessible to a lay user who is not adept at formal querying languages. With the current wide usage of linked data, NLQF is essential in bridging the human–computer divide. In this paper, an approach to creating an NLQF pipeline with Rasa, an open-source voice and text-based AI assistant development framework that acts as a semi-black box aspect, is discussed. Individual components such as intents, entities, actions, and stories were introduced and discussed in some detail, especially in the scope of this model along with the introduction

of in-built training modules and how they can be configured. The development of the ontology model of the college hostel system has also been touched upon. Finally, the validation of this NLQF pipeline is analysed and discussed in detail. Throughout the implementation of this model, some drawbacks have been identified which are as follows:

1. Scalability: The current model is context-dependent and is not scalable to very large knowledge sets such as DBpedia.
2. Compound queries: The model has a comparatively low recall for compound queries of the type where a query like: "What kind of rooms are available in the girl's hostel with the highest capacity." Instead, it needs to be broken into two queries like: "Which is the girl's hostel with the highest capacity" and "What kind of rooms are available in this hostel?" for more accurate results.
3. More comprehensive query type model: The model can benefit from the integration of a more comprehensive query type model, which classifies simple, complex, and compound queries further in terms of scalars, negations, comparatives, etc.

While the aforementioned points exist as substantial challenges, there will be work done in the future to overcome them. This pipeline will be made more scalable through query type classification using Rasa with a denser entity set to complement it while the problem of compound queries may be addressed by internal query segmentation and processing. The methodology discussed in this paper is being not used for commercial purposes but only for academic research.

Acknowledgements This work was supported by the Information and Communication Department at Manipal Institute of Technology whose professors provided us constant mentorship.

Author Contributions All authors contributed to the conception and design of this approach. Data generation, story creation, identification of intents and entities from the data and training of the model was done by Divyansh. Generation of ontology, development of querying scripts, and generation of validation dataset were done by Abhinav. First draft of the paper was written by Divyansh, and results were compiled by Abhinav. Swathi BP and Akshay KC provided their valuable guidance throughout the project and especially helped in improving subsequent drafts of the paper. All authors read and approved the final manuscript.

Funding Open access funding provided by Manipal Academy of Higher Education, Manipal. The authors declare that this work was performed without any internal or external funding in any shape or form.

Availability of data The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request

Declarations

Conflict of interest The authors declare that they have no competing interests.

Availability of code The code used for the approach discussed in this study is available from the corresponding author on reasonable request

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Bahmid, R., Zouaq, A.: Hybrid question answering using heuristic methods and linked data schema. In: 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), pp. 446–451. IEEE (2018)
2. Baudiš, P., Šedivý, J.: Modeling of the question answering task in the yodaqa system. In: International Conference of the Cross-Language Evaluation Forum for European Languages, pp. 222–228. Springer (2015)
3. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1247–1250 (2008)
4. Boumechaal, H., Boufaïda, Z.: Formalization of natural language queries. In: 2011 International Symposium on Innovations in Intelligent Systems and Applications, pp. 495–499. IEEE (2011)
5. Chen, D., Fisch, A., Weston, J., Bordes, A.: Reading wikipedia to answer open-domain questions. [arXiv:1704.00051](https://arxiv.org/abs/1704.00051) (2017)
6. Chen, Q., Zhu, X., Ling, Z., Wei, S., Jiang, H., Inkpen, D.: Enhanced lstm for natural language inference. [arXiv:1609.06038](https://arxiv.org/abs/1609.06038) (2016)
7. Demidova, E., Fankhauser, P., Zhou, X., Nejdl, W.: Divq: diversification for keyword search over structured databases. In: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 331–338 (2010)
8. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmman, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 601–610 (2014)
9. Dubey, M., Dasgupta, S., Sharma, A., Höffner, K., Lehmann, J.: Asknow: A framework for natural language query formalization in sparql. In: European Semantic Web Conference, pp. 300–316. Springer (2016)
10. Elbedweihy, K., Wrigley, S.N., Ciravegna, F.: Evaluating semantic search query approaches with expert and casual users. In: International Semantic Web Conference, pp. 274–286. Springer (2012)
11. Elbedweihy, K., Wrigley, S.N., Ciravegna, F., Reinhard, D., Bernstein, A.: Evaluating semantic search systems to identify future

- directions of research. In: Extended Semantic Web Conference, pp. 148–162. Springer (2012)
12. Fabian, M., Gjergji, K., Gerhard, W., et al.: Yago: A core of semantic knowledge unifying wordnet and wikipedia. In: 16th International World Wide Web Conference, WWW, pp. 697–706 (2007)
 13. Ferré, S.: Sparklis: an expressive query builder for sparql endpoints with guidance in natural language. *Semantic Web* **8**(3), 405–418 (2017)
 14. Habernal, I., Konopíková, M.: Swsnl: semantic web search using natural language. *Expert Syst. Appl.* **40**(9), 3649–3664 (2013)
 15. Hao, Y., Zhang, Y., Liu, K., He, S., Liu, Z., Wu, H., Zhao, J.: An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 221–231 (2017)
 16. He, H., Meng, W., Yu, C., Wu, Z.: Wise-integrator: a system for extracting and integrating complex web search interfaces of the deep web. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 1314–1317 (2005)
 17. Hu, X., Dang, D., Yao, Y., Ye, L.: Natural language aggregate query over rdf data. *Inf. Sci.* **454**, 363–381 (2018)
 18. Indurthi, S.R., Raghu, D., Khapra, M.M., Joshi, S.: Generating natural language question-answer pairs from a knowledge graph using a rnn based question generation model. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pp. 376–385 (2017)
 19. Iyer, S., Konstas, I., Cheung, A., Krishnamurthy, J., Zettlemoyer, L.: Learning a neural semantic parser from user feedback. [arXiv:1704.08760](https://arxiv.org/abs/1704.08760) (2017)
 20. Jagannatha, A.N., Yu, H.: Structured prediction models for rnn based sequence labeling in clinical text. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing, vol. 2016, p. 856. NIH Public Access (2016)
 21. Jiao, A.: An intelligent chatbot system based on entity extraction using rasa nlu and neural network. *JPhCS* **1487**(1), 012014 (2020)
 22. Karimi, E., Majidi, B., Manzuri, M.T.: Relevant question answering in community based networks using deep lstm neural networks. In: 2019 7th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), pp. 1–5. IEEE (2019)
 23. Kaufmann, E., Bernstein, A.: How useful are natural language interfaces to the semantic web for casual end-users? In: The Semantic Web, pp. 281–294. Springer (2007)
 24. Kim, J.D., Cohen, K.B.: Natural language query processing for sparql generation: a prototype system for snomed ct. In: Proceedings of biolink, vol. 32, p. 38 (2013)
 25. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al.: Dbpedia-a large-scale, multilingual knowledge base extracted from wikipedia. *Seman. web* **6**(2), 167–195 (2015)
 26. Li, F., Jagadish, H.V.: NaLIR: an interactive natural language interface for querying relational databases. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of data, pp. 709–712 (2014)
 27. Murayama, Y., Kobayashi, I., Morita, T., Nakano, Y.I., Yamaguchi, T.: Development of an access method to menu ontology data through converting natural language into sparql queries. In: JIST (Workshops & Posters), pp. 13–20 (2018)
 28. Ochieng, P.: Parot: Translating natural language to sparql. *Expert Syst. Appl.* **X** **5**, 10024 (2020)
 29. Shaik, S., Kanakam, P., Hussain, S.M., Suryanarayana, D.: Transforming natural language query to sparql for semantic information retrieval. *Int. J. Eng. Trends Technol.* **41**, 347–350 (2016)
 30. Sun, C., Qiu, X., Xu, Y., Huang, X.: How to fine-tune bert for text classification? (2019)
 31. Tatu, M., Balakrishna, M., Werner, S., Erekhinskaya, T., Moldovan, D.: Automatic extraction of actionable knowledge. In: 2016 IEEE Tenth International Conference on Semantic Computing (ICSC), pp. 396–399 (2016)
 32. Wilson, M.L., Kules, B., Schraefel, M., Shneiderman, B.: From Keyword Search to Exploration: Designing Future Search Interfaces for the Web, vol. 1. Now Publishers Inc, New York (2010)
 33. Xu, X., Liu, C., Song, D.: Sqlnet: Generating structured queries from natural language without reinforcement learning. [arXiv:1711.04436](https://arxiv.org/abs/1711.04436) (2017)
 34. Yaghmazadeh, N., Wang, Y., Dillig, I., Dillig, T.: Sqlizer: query synthesis from natural language. *Proc. ACM Program. Lang.* **1**(OOPSLA), 1–26 (2017)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.