Constructing a KBQA Framework: Design and Implementation

Rômulo Chrispim de Mello
romulomello@ice.ufjf.br
Department of Computer Science
Federal University of Juiz de Fora
Juiz de Fora – MG, Brazil

Jorão Gomes Jr.
jorao.gomes.junior@wu.ac.at
Institute for Digital Ecosystems
Vienna University of Economics and Business
Vienna, Austria

Jairo Francisco de Souza
jairo.souza@ice.ufjf.br
Department of Computer Science
Federal University of Juiz de Fora
Juiz de Fora – MG, Brazil

Victor Ströele
victor.stroele@ice.ufjf.br
Department of Computer Science
Federal University of Juiz de Fora
Juiz de Fora – MG, Brazil

ABSTRACT

The exponential growth of data on the internet has made information retrieval increasingly challenging. Knowledge-based Question-Answering (KBQA) framework offers an efficient solution that quickly provides accurate and relevant information. However, these frameworks face significant challenges, especially when dealing with complex queries involving multiple entities and properties. This paper studies KBQA frameworks, focusing on improving entity recognition, property extraction, and query generation using advanced Natural Language Processing (NLP) and Artificial Intelligence (AI) techniques. We implemented and evaluated combination tools for extracting entities and properties, with the combination of models achieving the best performance. Our evaluation metrics included entity and property retrieval, SPARQL query completeness, and accuracy. The results demonstrated the effectiveness of our approach, with high accuracy rates in identifying entities and properties.

KEYWORDS

KBQA, Complex Questions, Entity Recognition, Property Extraction, SPARQL

1 INTRODUCTION

The large volume of data available on the internet has made the task of finding relevant information even more challenging [11]. In this context, new information retrieval methods have allowed for more intelligent searches, taking into account the context of the search. Additionally, advances in Natural Language Processing (NLP) research have enabled a better understanding of the search context, allowing Knowledge-based Question-Answering (KBQA) frameworks to emerge as an efficient solution to meet this demand [12].

The complexity of the questions users ask is one of the main challenges faced by KBQA frameworks [17]. Questions can be complex for various reasons, such as the presence of multiple concepts, the need to understand the context in which the question is created, and the dependence on additional information that was not mentioned in the question. These factors can lead to ambiguities in questions and, consequently, to imprecise results.

Natural language questions can vary greatly in complexity. Simple questions might involve straightforward retrieval of facts, such as "What is the capital of France?". However, complex questions often require the integration of multiple pieces of information, reasoning over data, and understanding nuanced context. For example, the question "Which films directed by Quentin Tarantino were nominated for an Oscar?" requires the system to identify multiple entities ("Quentin Tarantino", "films", and "Oscar") and understand their relationships [23]. The complexity increases when questions involve conditional statements, comparative structures, or temporal aspects. For instance, "What was the population of New York City before 2000?" needs temporal reasoning and access to historical data. Similarly, questions like "Is the Eiffel Tower taller than the Statue of Liberty?" require comparative reasoning and precise entity linking.

Understanding the context in which a question is asked is crucial for accurate KBQA. The same term can have different meanings based on context, making disambiguation a significant challenge. For example, the word "Java" can refer to a programming language, an island in Indonesia, or a type of coffee. Determining the correct interpretation based on the surrounding context is essential for providing accurate answers [10]. Moreover, questions often rely on implicit context that is not explicitly stated. For instance, in the question "What is the country's capital?", the system must infer which country is being referred to from prior context or user interaction history. This requires the KBQA system to maintain and utilize contextual information dynamically.

Large language models (LLMs) such as GPT have recently gained popularity as Question-Answering (QA) systems, demonstrating impressive results in answering questions posed in natural language. These models are trained on massive amounts of text data, allowing them to understand and generate human-like responses to various prompts [2]. However, it is important to note that QA systems solely using LLMs are limited by their training data [20]. They are not designed to access external knowledge repositories and cannot provide information not included in their training data. For instance, if a question is asked about an event that occurred after

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia'2024). Juiz de Fora, Brazil. Porto Alegre: Brazilian Computer Society, 2024.

ISSN 2966-2753

the LLM's training data was collected, it may be unable to provide an accurate answer.

On the other hand, KBQA systems are designed to access and search knowledge repositories, providing access to always up-to-date information [14]. These systems can retrieve information from various structured and unstructured data sources, including databases and ontologies [4]. With their ability to access a large and constantly updated knowledge base [15], KBQA systems are more suitable for answering complex questions that require domain-specific knowledge or information about recent events.

Complex Knowledge-based Question-Answering (C-KBQA) systems use natural language processing and artificial intelligence techniques [8]. Natural language processing allows the system to understand and interpret users' questions, while artificial intelligence enables the system to learn the characteristics of those questions, their entities, properties, and relationships and to return the appropriate values. Additionally, these systems have a robust and up-to-date knowledge base, which allows the retrieval and presentation of relevant and current information to users [15].

KBQA systems typically involve many steps in query processing, from entity recognition to SPARQL query generation. In this work, we propose a KBQA framework that features a practical and flexible pipeline for complex knowledge-based question answering (KBQA). This framework can serve as a model for other systems of this type, including adaptations to other languages.

The proposed pipeline is designed to handle complex questions involving multiple entities and properties and generate accurate and complete SPARQL queries. A central feature of this pipeline is its flexibility, allowing the replacement of entity recognition and property extraction models with new ones that may offer better performance or accuracy. Additionally, the pipeline can incorporate refinements in SPARQL queries, using qualifier constraints for specific properties, increasing the completeness and accuracy of the answers. We implemented methods to retrieve all properties related to an entity and developed a system to dynamically fill placeholders in templates with the correct values extracted from the knowledge base. These methods ensure the pipeline can easily adapt to new requirements and improvements, maintaining its effectiveness and efficiency.

This structured approach ensures the system can handle complex queries, decompose them, identify relevant entities and relationships, and construct precise SPARQL queries to provide accurate results. Using entity recognition, relation extraction, and template matching, combined with effective ranking and slot filling, enhances the accuracy of knowledge-based question answering. By structuring the pipeline in a modular and flexible way, it can easily integrate new models and rules, ensuring continuous improvements in accuracy and efficiency. Thus, the pipeline demonstrates how to handle complex queries effectively and provides a solid foundation for future adaptations and innovations in the field of KBQA systems.

This paper is organized as follows: Section 2 presents related work, discussing existing approaches and their contributions to the field. Section 3 describes the materials and methods used in this study, including dataset preprocessing, template grouping by semantic proximity, dummy template creation, and the tools used for entity and property extraction. Section 4 presents and discusses the obtained results, highlighting the tool combinations that achieved the best performance. Section 5 concludes the work, discussing the implications of the results and suggesting future research directions.

2 RELATED WORK

Research in Knowledge-Based Question Answering (KBQA) has explored various approaches to enhance the precision and efficiency of these systems, particularly when dealing with complex questions. This section reviews significant contributions across three main areas: entity recognition, relation extraction, and template matching.

2.1 Entity Recognition

Entity recognition is a fundamental task in KBQA, involving identifying entities mentioned in the user's query. One notable work in this area is TagMe [6], which introduces a technique for annotating short text fragments with relevant Wikipedia hyperlinks. The system leverages Wikipedia's extensive collection of articles and anchor texts to provide informative and accurate annotations. The main technique used involves point identification, sense disambiguation, and annotation. Point identification analyzes the input text to identify potential points that can be linked to Wikipedia articles. Sense disambiguation selects each point's most relevant Wikipedia page, considering context and statistical information. Finally, annotation adds hyperlinks to corresponding Wikipedia articles, allowing users to access additional information and context simply by clicking on the annotated points. TagMe demonstrates superior performance and speed compared to other systems, making it a valuable tool for entity recognition in short texts.

Another significant contribution in this field is Falcon [19], a rule-based approach for linking entities and relationships in Wikidata. Falcon employs core principles of English morphology, such as tokenization and N-gram tessellation, to link entity and relation surface forms in short sentences to Wikidata entries. This method includes a local knowledge base composed of DBpedia entities to enhance the recognition and linking process. Falcon provides a ranked list of entities and relations annotated with their Internationalized Resource Identifier (IRI) in Wikidata, aiding the NLP community in entity and relation recognition. The approach outperforms existing baselines in entity linking tasks, demonstrating high F-score values and robustness across various datasets like QALD-9 [16] and LC-QuAD 2.0 [5].

2.2 Relation Extraction

Relation extraction is another crucial component of KBQA systems, focusing on identifying and linking relationships between entities within a query. SLING [13] is a semantic analysis framework designed to link text relationships to knowledge bases accurately. The approach integrates multiple methods, including statistical Abstract Meaning Representation (AMR) mapping, distant supervision data generation, and various relation-linking modules. The statistical AMR mapping technique is pivotal in identifying relationships by normalizing syntactic variations between sentences and providing strong predicates. Distant supervision data generation creates training examples mapped to corresponding knowledge base relations, enhancing the system's learning process. SLING leverages

transformer-based architectures to encode AMR graphs and question text for relation linking, achieving state-of-the-art performance across datasets such as QALD-7 [22], QALD-9 [16], and LC-QuAD 1.0 [21].

Another innovative approach in relation extraction is presented by [18], which proposes a sequence-to-sequence model enhanced with structured data from the target knowledge base. This model generates a sequence of relations based on the input question text, enriched by an entity-linking system that queries the knowledge base to retrieve candidate relations. The model's decoder then uses the enriched input representation to generate a structured sequence of argument-relation pairs, considering the contextual information and candidate relations. This approach significantly improves relation-linking performance in question-answering systems, demonstrating notable enhancements over existing methods.

2.3 Template Matching

Template matching involves identifying patterns in user questions and matching them to predefined templates, facilitating the generation of structured and well-formatted responses. In [3], the authors explore machine learning models and preprocessing techniques to classify natural language questions into appropriate templates using the LC-QUAD 2.0 dataset. They train classifiers such as XGBoost and Random Forest, utilizing Part-of-Speech (POS) tagging and FastText for preprocessing. POS tagging assigns grammatical tags to words, helping the system understand the syntactic structure of questions, while FastText captures the semantic meaning of words through embeddings. The combination of XGBoost and POS+FastText preprocessing achieves superior accuracy in classifying questions into relevant templates, showcasing the effectiveness of template-based question answering.

Another noteworthy study is presented by [7], which introduces a hereditary attention mechanism combined with template matching to enhance semantic extraction from questions. This approach categorizes complex questions into answer templates, leveraging hierarchical structures within the questions. The hereditary attention mechanism operates bottom-up, where each neural network cell inherits attention from another cell, capturing and prioritizing the most relevant information at different levels of the question's structure. This method improves the robustness and accuracy of KBQA systems, providing a reliable technique for answering complex questions from knowledge bases.

2.4 Frameworks

Recent advancements in KBQA framework have highlighted the effectiveness of integrating various methodologies to improve performance in complex queries. Two works in this area are the RnG-KBQA and ReTraCk framework, which present approaches to enhance the accuracy and generalization capabilities of KBQA framework.

The RnG-KBQA system, developed by [24], addresses the limitations of traditional KBQA systems that struggle with unseen knowledge base (KB) schema items. RnG-KBQA combines a ranking-based approach with a generation model to enhance coverage and generalization. The system first ranks candidate logical forms based on the question. Then, it uses a generation model conditioned on the question and top-ranked candidates to create the final logical form. This dual approach significantly improves performance, achieving state-of-the-art results on the GRAILQA and WEBQSP datasets, with notable improvements in zero-shot generalization.

However, RnG-KBQA also presents some limitations. The complexity of combining ranking and generation can lead to longer processing times and increased computational resource requirements. Additionally, the effectiveness of the generation model may be limited by the quality of the training data, affecting the system's ability to handle ambiguous or poorly formulated queries.

The ReTraCk, developed by [1], introduces a flexible framework that integrates multiple stages for entity recognition, relation extraction, and ranking. ReTraCk emphasizes the importance of a modular design, allowing for easy integration of different models and techniques at each stage. The system has shown remarkable performance in various benchmarks, demonstrating its adaptability and efficiency in handling diverse KBQA tasks. The approach focuses on iterative refinement and ranking to enhance the accuracy of the generated answers, contributing to the development of a more robust KBQA framework.

Despite its advantages, ReTraCk also faces challenges. Integrating and adjusting multiple models and techniques can increase the system's complexity, making maintenance and updates more difficult. Additionally, since ReTraCk relies on multiple processing stages, any error in one of these stages can compromise the accuracy of the final answer. The modular approach, while flexible, can result in inconsistencies when different modules are not perfectly aligned. Moreover, ReTraCk is a resource-intensive system requiring considerable computational resources, which can be a barrier to deployment in production environments with limited resources.

These works underscore the importance of combining ranking and generation techniques to overcome the limitations of the traditional KBQA framework, paving the way for more accurate and generalizable solutions. By integrating advanced NLP and machine learning models, both RnG-KBQA and ReTraCk contribute significantly to the field, offering valuable insights and methodologies for future research in KBQA. However, the identified limitations of these systems highlight the ongoing need for refinement and innovation to improve the efficiency and effectiveness of KBQA systems.

This work shares the same principles of these frameworks, such as flexibility and their use for different datasets. However, our approach stands out by focusing on complex queries, which often require the integration of multiple pieces of information, reasoning over data, and understanding nuanced context. The solution allows new entity recognition and property extraction models if needed. We implemented methods to retrieve all properties related to an entity and developed a system to dynamically fill placeholders in templates with the correct values extracted from the knowledge base. These methods ensure the pipeline can easily adapt to new requirements and improvements. Finally, the system's structured design can decompose complex queries, identify relevant entities and relationships, and construct precise SPARQL queries using entity recognition, relation extraction, template matching, ranking, and slot-filling methods.

3 MATERIALS AND METHOD

To validate our complex question-answering framework, we used the LC-QUAD 2.1 dataset [9], a refined and cleaned version of LC-QUAD 2.0. LC-QUAD 2.1 was chosen due to its structure, which includes question templates and SPARQL queries with "dummy" elements, facilitating subsequent filling with specific information. This dataset was created based on the improvements made by [7].

The LC-QUAD 2.0 dataset [5] was created using the Amazon Mechanical Turk crowdsourcing platform, resulting in over 30,000 complex questions intended to be answered by knowledge-based question-answering (KBQA) systems. However, the original dataset presented several inconsistencies, such as duplicate questions, malformed questions, and extreme variations in question length. To address these issues, [7] developed LC-QUAD 2.1. This new dataset underwent a cleaning and standardization process, removing duplicate and malformed questions and adjusting question lengths to ensure greater consistency. This process resulted in a cleaner and more reliable dataset for testing KBQA systems.

The questions in this dataset are complex due to their multifaceted nature, involving multiple entities, relationships, and advanced operations. These questions often require combining data from different sources, applying temporal filters, and specific counting operations. Additionally, many questions use qualifiers to provide more detailed, contextually rich answers. Complexity also includes the need for context disambiguation and handling multiple intents in a single question. It also covers various question types, including Boolean questions, counting operations, and questions requiring string operations and temporal aspects. This diversity and complexity are crucial for evaluating the robustness and effectiveness of KBQA systems, ensuring they can handle realistic and challenging queries.

We then started working on grouping templates by semantic proximity. This process involved identifying common elements in SPARQL queries, such as projections, jumps, filters, and logical operators, and organizing them into coherent groups based on their complexity and structure. Projections define the specific data to be returned, while jumps represent the intermediate steps needed to obtain that data by navigating relationships within the knowledge graph. Filters narrow down results based on criteria like numeric comparisons or string matches, and logical operators such as LIMIT, ORDER BY, COUNT, and DISTINCT modify the presentation of results.

To group the templates, we applied NLP techniques to analyze the relationships between these components and measure their semantic similarities. This allowed us to categorize the templates into groups that reflect both their structural and semantic characteristics. For example, Group 0 includes simple ASK queries that verify the existence of relationships, while Group 1 also uses ASK queries but involves multiple jumps between entities to verify more complex relationships. Group 2 focuses on selecting distinct results. As the complexity increases, groups such as 3 through 7 involve counting entities, applying filters, and ordering results. The more advanced groups, 8 through 12, handle multiple relationships between entities, require complex string matching, temporal filtering, and often combine multiple values and nested conditions, significantly increasing query complexity.

These grouped templates ensure that similar queries are organized in a way that preserves their logical structure, facilitating efficient retrieval and adaptation for different questions. Empirical validation using real datasets confirmed the effectiveness of these groupings, resulting in improved template retrieval at runtime, allowing the system to quickly and accurately handle a wide range of complex questions.

Table 1: Examples of Grouped Templates

Group	ID	SPARQL Template
0	0.1	`ASK WHERE { DUMMY_S DUMMY_P DUMMY_O }`
	0.2	`ASK WHERE { DUMMY_S DUMMY_P ?obj FILTER(?obj = DUMMY_F) }`
	0.3	`ASK WHERE { DUMMY_S DUMMY_P ?obj FILTER(?obj > DUMMY_F) }`
	0.4	`ASK WHERE { DUMMY_S DUMMY_P ?obj FILTER(?obj < DUMMY_F) }`
2	2.1	`SELECT DISTINCT ?answer WHERE { ?answer DUMMY_P DUMMY_O }`
	2.2	`SELECT DISTINCT ?answer WHERE { DUMMY_S DUMMY_P ?answer }`

As shown in Table 1, these examples illustrate how templates with similar structures and semantic functions are grouped to facilitate retrieval and adaptation. By grouping templates in this way, the system can reuse and adapt predefined templates more efficiently, answering a wide range of complex questions accurately and quickly.

The creation of dummy templates was another fundamental aspect of the work. These templates are generic SPARQL queries where placeholders, such as DUMMY_S for the subject, DUMMY_P for the predicate, and DUMMY_O for the object replace specific elements. Analyzing existing SPARQL queries allowed the identification of common patterns, which were generalized into representative dummy templates. These templates were then grouped based on semantic similarities, using NLP techniques to organize the templates efficiently.

The dummy elements in LC-QUAD 2.1 are essential for our framework because they allow a structured and systematic slot-filling process while generating the final SPARQL queries. These dummies are replaced by actual values after the correct entities, properties, and filters have been identified and extracted. Moreover, queries with dummies are retrieved through models trained by [7] and [3]. These models were developed to identify and structure complex questions, facilitating the creation of SPARQL queries that use dummies to represent entities and properties to be filled later.

To use this dataset in the framework, we performed initial processing starting from the gold SPARQL query for each question. We extracted entities, properties, and filters. At the end of the pipeline processing, this extracted information is used to evaluate the system, checking whether all expected entities, properties, and filters or only a portion were found. This allows us to measure the precision and efficiency of the framework by comparing the expected information, ensuring that the system is aligned with the gold SPARQL query.

3.1 Tools for Entity and Property Extraction

With information about the templates, entities, and properties needed to answer each question, we began searching for tools that could efficiently perform this extraction. We identified three main tools: DeepPavlov, Falcon 2.0, and Spacy. Each of these tools has unique features that contribute to the accuracy and coverage of our system.

DeepPavlov is an open-source NLP library that offers a variety of pre-trained models for different NLP tasks, including named entity recognition (NER) and entity linking. The DeepPavlov library is known for its flexibility and efficiency, allowing customization of models as needed.

Specifically, we used the Entity Linking model from DeepPavlov. This model operates in two main stages. The first is named entity recognition (NER), where the model identifies entities in the text using a combination of linguistic rules and deep learning models. It applies tokenization techniques, where the text is divided into tokens, which are analyzed to identify possible entities. Then, entity disambiguation, where the Entity Linking model links these entities to specific entries in a knowledge base, such as Wikidata. This process uses word embeddings, which are word representation vectors, to calculate similarities between the identified entities in the text and the candidates in the knowledge base, selecting the best match.

Falcon 2.0 links entities and relations in short texts specifically designed to work with Wikidata. According to [19], Falcon 2.0 uses a rule-based linguistic approach. The tool performs tokenization and compounding, dividing the text into tokens using English morphology rules. Tokens representing entities or relations are identified, and compound tokens (like "operating income") are treated as a single unit. It also uses N-Gram tiling techniques to combine tokens close to the text and can form a composite entity or relation. When necessary, the model can also split N-Grams to refine the identification of entities and relations.

Falcon 2.0 generates a list of candidate entities and possible relations from Wikidata for each identified token. This is done by consulting a local knowledge base that contains alignments of labels and aliases of entities and relations from Wikidata. The generated candidates are then ranked based on similarity and the probability of a correct match. The model uses rules and an inference process to determine the most likely candidates. Falcon 2.0 stands out for its ability to simultaneously link entities and relations, which is crucial for answering complex questions involving multiple entities and their interrelations. The tool is available as an online API, making it accessible and easy to integrate into our system.

Finally, Spacy is a widely used open-source NLP library known for its speed and efficiency. It offers robust models for tasks such as named entity recognition, dependency parsing, and text classification. We used Spacy's entity linking model. The text is processed through a pipeline that includes tokenization, lemmatization, POS tagging (part-of-speech tagging), and dependency parsing. Each of these steps contributes to accurately identifying entities in the text.

Spacy uses a deep learning model trained on large annotated datasets to identify entities in the text. This model recognizes entities in various contexts and domains. After identifying the entities, the entity linking model associates them with a knowledge base, such as Wikidata. Spacy uses entity embeddings and a vector similarity approach to find the best match for each identified entity. The library allows significant customizations, enabling adjustments to model parameters and adding new entities and relations to the knowledge base as needed.

3.2 Tools Integration

After selecting the tools, we extracted entities and properties from each question using the described tools. Each tool was applied independently, and the results were compared to identify complementarities and redundancies. This process was fundamental to ensuring that all relevant entities and properties were captured, increasing the accuracy and coverage of our system.

3.2.1 Additional Properties Retrieval. We noticed that many properties were still not captured despite using these tools. To address this, we implemented a method to retrieve all properties related to an entity, treating it both as a subject and as an object.

To retrieve these additional properties, we performed specific SPARQL queries. When the entity was treated as a subject, the query searched for all properties where the entity was the subject of the relationship. Conversely, when the entity was treated as an object, the query searched for all properties where the entity was the object of the relationship. These querying processes allowed us to recover a broader set of properties related to the identified entities.

The queries were executed for each identified entity, and the results returned the related properties and descriptions. These additional properties were then integrated into the original dataset, improving the coverage and accuracy of our system. This additional process allowed us to expand the set of captured properties, enhancing the comprehensiveness and precision of our system. However, there was still a type of property that we could not accurately capture.

3.2.2 Qualifiers Constraints for Specific Properties. Our system could not generate correct answers for specific templates even after using the tools and retrieving additional properties. We implemented an additional approach to address these cases using qualifier constraints of specific properties. Qualifiers are metadata that add context to a statement in a knowledge base like Wikidata, specifying additional conditions that must be met. These qualifiers can be essential for correctly understanding the application of certain properties in specific contexts.

To identify and use these qualifiers, we developed specific SPARQL queries. These queries searched for the constraints associated with particular properties, allowing us to refine data extraction further and improve the accuracy of the answers generated by our system. The query was structured to identify "qualifier value" constraints associated with a specific property. This query selects objects and their labels associated with the specified property's constraints.

We ran this query for problematic properties identified during the query generation process. By retrieving these constraints, we could identify the additional conditions needed for the correct property application. These qualifier constraints significantly improved

our system's ability to generate accurate answers for the problematic templates. This allowed us to overcome previous limitations and increase the framework's overall precision.

3.2.3 Ranking. Although we had a relatively small number of entities, the number of properties was very large. This made it difficult to combine entities with properties, resulting in many possible SPARQL queries. To address this problem, we used the relation ranking module of DeepPavlov to filter and select the most relevant properties for each question.

The relation ranking module of DeepPavlov is designed to take a question and a set of properties as input and then rank them based on their relevance using a ranking model. The module assigns a specific probability to each property, allowing the filtering of the most relevant ones for the given question. This process is essential to reduce the complexity of slot filling, where combining all entities with all possible properties would result in a huge number of queries.

The relation ranking process uses word embeddings to represent words in vector space semantically. These embeddings are loaded from pre-trained models such as fastText, used in our case. The ranking module receives the set of previously extracted properties for each question. It processes these properties along with the question and calculates the probabilities of the properties being relevant to the question. The softmax layer in the model is responsible for assigning probabilities, allowing only the most probable properties to be selected. Thus, for each question, we selected the top 5 properties from the ranking. This filtered the entities, keeping only those related to the selected properties. This process reduced the number of properties to be considered and restricted the relevant entities, making the combination of entities and properties more manageable and precise. This approach was necessary to reduce the complexity of slot filling and avoid generating excessive SPARQL queries. By focusing on the most relevant properties and entities, we improved efficiency, ensuring the system could adequately respond to complex questions.

3.2.4 Filters. In addition to extracting entities and properties, a crucial step in our system was the extraction of filters. Filters are additional components of SPARQL queries that restrict results based on specific criteria, such as integer values, dates, or other conditions. Identifying these filters is essential for generating correct SPARQL queries.

Since we already knew which templates contained filters, we developed rules based on regular expressions (regex) specific to each template type. Each set of filters was associated with these specific templates. Regex are tools used to identify specific patterns within texts. First, we analyzed the templates to identify common filter patterns, such as dates and numbers. Then, we defined specific regex to capture these values in each template type and implemented these rules in the processing pipeline. For example, we used regex to identify dates or integer numbers in questions that contained these as filters. We tested these rules on a validation set to ensure the correct extraction of filters and adjusted the regex as necessary to handle variations in question patterns.

After extraction, the filters were integrated into the SPARQL query processing pipeline. Each filter was associated with the corresponding entities and properties within the context of the specific templates, completing the query construction. This process allowed our system to handle complex questions with specific conditions, ensuring that the generated SPARQL queries were accurate. The regex-based approach specific to each template type proved efficient in identifying and extracting filters directly from the questions, improving the system's ability to accurately respond to a wide range of questions.

3.3 Pipeline

Our KBQA system was designed to efficiently handle complex queries by breaking the process into several stages. Figure 1 depicts the complete workflow for answering the query: "Which films directed by Quentin Tarantino were nominated for an Oscar?"

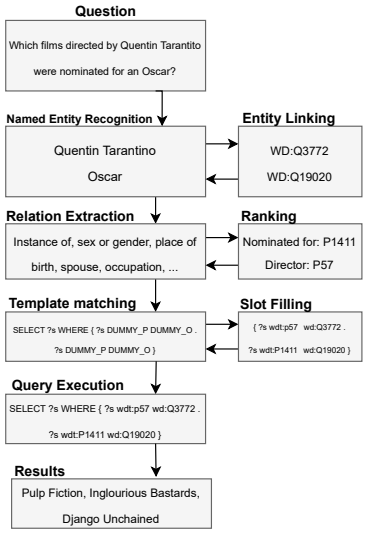

Figure 1: Workflow of the KBQA system for the question: "Which films directed by Quentin Tarantino were nominated for an Oscar?"

Initially, the input question is parsed to understand its structure and intent. Key entities such as "Quentin Tarantino" and "Oscar" are recognized through Named Entity Recognition (NER) and subsequently linked to their corresponding entries in the knowledge base (in our case, the Wikidata). Relationships between these entities and other relevant attributes are then extracted. These entities and relations are ranked based on their relevance to the query, prioritizing properties like "nominated for" (P1411) and "director" (P57).

Next, the query is matched to a predefined template to help generate the appropriate SPARQL query. Specific slots in the SPARQL query are filled with the identified entities and properties. The completed SPARQL query is executed against the knowledge base, retrieving the relevant data. For instance, the query *SELECT ?s WHERE ?s wdt:P57 wd:Q3772 . ?s wdt:P1411 wd:Q19020* fetches films directed by Quentin Tarantino and nominated for an Oscar. The results are then processed and presented, identifying films like "Pulp Fiction", "Inglourious Basterds", and "Django Unchained".

This structured approach ensures the system can handle complex queries by breaking down the question, identifying relevant entities and relations, and constructing precise SPARQL queries to provide accurate results. Using entity recognition, relation extraction, and template matching, combined with effective ranking and slot filling, enhances the accuracy of knowledge-based question answering.

The central idea of our framework is to provide robust flexibility, allowing the insertion and removal of models and rules as needed. This includes replacing existing entity recognition systems with new, potentially more accurate models, and allowing multiple systems to operate in parallel for greater accuracy. The property extraction component is tunable, allowing you to use different models or updated rules to refine SPARQL queries. The classification system can be modified in some parameters or completely replaced to prioritize the most relevant entities and properties.

This flexibility is crucial for adapting the system to diverse needs and contexts, enabling continuous improvements in accuracy and efficiency. By allowing you to add and remove models, with advances in NLP and AI techniques, the framework ensures it can handle complex queries, identify relevant entities and relationships, and build accurate SPARQL queries to provide accurate results. This structured approach increases the accuracy of answers to knowledge-based questions.

3.4 Metrics Description

To evaluate the effectiveness of our KBQA system, we used three metrics that allow us to measure both the precision and coverage of the answers provided: Correct Entities, Correct Properties, and Corret SPARQL. The **Correct Entities** are the number of entities correctly identified in the queries. Identifying entities accurately is crucial because entities represent the key components of a query. For example, in the query "Which films directed by Quentin Tarantino were nominated for an Oscar?", "Quentin Tarantino" and "Oscar" are entities that need to be correctly identified. The **Correct Properties** are the number of properties correctly identified in the queries. Properties describe the relationships or attributes of entities. For example, in the query "Which films directed by Quentin Tarantino were nominated for an Oscar?", "directed by" and "nominated for" are properties that must be identified accurately. Finally, the **Correct SPARQL** is the percentage of queries where the SPARQL query was generated correctly. This metric is critical as it reflects the overall ability of the system to understand and translate natural language queries into correct SPARQL queries, which are necessary for retrieving accurate information from knowledge bases.

These metrics are essential for a comprehensive evaluation of the KBQA system. They provide a detailed breakdown of where the system performs well and where there are gaps. For example, high entity accuracy but low property accuracy would suggest that while the system is good at recognizing subjects, it struggles with understanding the relationships between them.

By analyzing metrics such as correct entities and properties, we can identify specific areas where the system needs improvement. For example, if many properties are incorrect, we might need to enhance our property extraction algorithms. Consider the query "Which films directed by Quentin Tarantino were nominated for an Oscar?". This query involves identifying the entity "Quentin Tarantino" and the properties "directed by" and "nominated for". The SPARQL query generated from this should accurately reflect these components to retrieve the correct films from the knowledge base.

Table 2: Correct Entities, Properties, and SPARQL queries found

Group	Entities (%)	Properties (%)	SPARQL (%)
0	82.32	67.09	62.94
1	84.92	86.67	51.67
2	76.66	52.47	37.14
3	94.30	74.68	73.42
4	65.01	54.92	14.29
5	81.76	55.97	14.32
6	42.62	24.42	0.00
7	96.76	51.18	10.03
8	97.50	84.38	76.25
9	53.96	7.17	2.26
10	95.06	66.67	30.86
11	55.06	32.91	0.00
12	84.46	50.90	14.86
Average	**77.72**	**54.57**	**29.85**
Average w/ filter	**85.87**	**64.49**	**38.58**

```
**Dummy Query:**
SELECT ?ent WHERE { ?ent DUMMY_P DUMMY_O . ?ent DUMMY_P ?obj
      } LIMIT DUMMY_F


**Complete SPARQL Query:**
SELECT ?film WHERE {
  ?film wdt:P57 wd:Q3772. # P57 represents "directed by" and
        Q3772 is Quentin Tarantino
  ?film wdt:P1411 wd:Q19020. # P1411 represents "nominated
       for" and Q19020 is Oscar
} LIMIT 10
```

In this example, if the entities "Quentin Tarantino" and "Oscar" are not identified correctly, or if the properties "directed by" and "nominated for" are not mapped correctly to their SPARQL representations, the query will fail to return the correct answer.

4 RESULTS AND DISCUSSION

Table 2 presents detailed results for the different tested tool combinations, highlighting the combination that showed the best performance: DeepPavlov, Falcon, and SpaCy.

The combination of DeepPavlov, Falcon, and SpaCy was selected after testing with other tools, which generally did not yield good results. These three tools proved to be the most efficient, complementing each other where one might fail individually. Other options were discarded due to recurring issues in achieving the necessary accuracy. Thus, this combination ensured a more reliable and balanced performance. The following sections discuss the main results observed.

4.1 Entity Recognition

Entity recognition is crucial for the accuracy of KBQA queries. The combination of tools was highly effective in identifying entities, achieving an entity accuracy rate of up to 95.06% in some template groups. This high rate of correct entity identification demonstrates the robustness of the tools in handling diverse and complex queries. The average entity accuracy rate was 77.72%, and when filtering the groups with the greatest difficulty (groups 6, 9, and 11), it increased to 85.87%. However, there is still room for improvement in reducing the number of missing entities, which can further enhance the overall framework performance.

4.2 Property Extraction

Property extraction showed strong performance, with accuracy rates reaching up to 86.67%. Accurately extracting properties is essential for generating precise and relevant answers to user queries. However, the average property accuracy rate (54.57%) was lower than that of entities (77.72%), which can be explained because property extraction depends on the correct identification of entities. If the entity is incorrect, the property will likely be incorrect. By filtering the groups with the greatest difficulty, the average property accuracy rate increases to 64.49%. The results indicate that the tools effectively understand and extract the required properties from queries. Continuous refinement of property extraction models will be beneficial in addressing this gap.

4.3 Overall Performance

The overall performance of the KBQA framework, reflected in the SPARQL accuracy rates, varied among different template groups. The highest SPARQL accuracy achieved was 76.25%, indicating that the system can generate highly accurate SPARQL queries under optimal conditions. The overall average SPARQL accuracy was 29.85%, and when filtering the groups with the greatest difficulty, this average increases to 38.58%. This demonstrates the potential of the tools to provide reliable and precise answers to complex queries. However, further enhancements in entity and property extraction will be critical to achieving consistently high performance across all queries.

4.4 Discussion

The results obtained have significant implications for the development of the KBQA framework. The 0 accuracy in some groups reflects the difficulty of handling complex questions, where multiple contextual meanings, intricate relationships between entities, and the need to infer implicit information make the task challenging. Improvements in context disambiguation and entity extraction are essential, as incorrect identifications lead to inaccurate answers. These challenges explain why other works often avoid complex questions, focusing on simpler issues where existing tools are more effective. Additionally, integrating inference mechanisms is crucial to capture implicit contextual information, providing more complete answers. Finally, enhancing scalability and performance is vital to ensure that the framework can handle large volumes of data.

Although the combination of DeepPavlov, Falcon, and SpaCy has shown promising results, there is still much room for improvement. So far, no fine-tuning has been performed on the models used, but this practice could bring significant improvements, especially in the groups where we encountered failures. The analysis of the results highlights key areas for future advancements, which will be essential for dealing with complex questions and advancing knowledge-based question-answering frameworks.

5 FINAL REMARKS

This work addresses the challenges and advancements in developing a practical and flexible pipeline for KBQA. We proposed a pipeline that can serve as a model for other KBQA systems, including adaptations to other languages. The pipeline is designed to handle complex questions involving multiple entities and properties, generating accurate and complete SPARQL queries.

The pipeline allows entity recognition and property extraction models to be replaced with new ones that may offer better performance or accuracy. We also implemented refinements in SPARQL queries, using qualifier constraints for specific properties, increasing the completeness and accuracy of the answers. Additionally, we developed methods to retrieve all properties related to an entity and dynamically fill placeholders in templates with the correct values extracted from the knowledge base.

Our structured approach ensures the system can handle complex queries, decompose them, identify relevant entities and relationships, and construct precise SPARQL queries to provide accurate results. Entity recognition, relation extraction, and template matching, combined with effective ranking and slot filling, have increased the accuracy of knowledge-based question answering.

The results highlight the effectiveness of the DeepPavlov, Falcon, and SpaCy combination, which demonstrated superior performance in identifying and extracting entities and properties. However, there is still room for continuous improvement, especially in reducing the number of unidentified entities and properties. Future research should improve context disambiguation techniques, enhance property extraction models, refine SPARQL queries for properties and qualifiers, and integrate more advanced inference mechanisms.

By structuring the pipeline in a modular and flexible way, it can easily integrate new models and rules, ensuring continuous improvements in accuracy and efficiency. Thus, the pipeline not only demonstrates how to effectively handle complex queries but also provides a solid foundation for future adaptations and innovations in the field of KBQA systems. Integrating KBQA with other emerging technologies, such as the Internet of Things (IoT), can open new possibilities for practical applications, further increasing the applicability and effectiveness of these systems.

REFERENCES

[1] Shuang Chen, Qian Liu, Zhiwei Yu, Chin-Yew Lin, Jian-Guang Lou, and Feng Jiang. 2021. ReTraCk: A flexible and efficient framework for knowledge base question answering. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing: system demonstrations*. 325–336.

[2] Xavier Daull, Patrice Bellot, Emmanuel Bruno, Vincent Martin, and Elisabeth Murisasco. 2023. Complex QA and language models hybrid architectures, Survey. *arXiv preprint arXiv:2302.09051* (2023).

[3] Akshay Kumar Dileep, Anurag Mishra, Ria Mehta, Siddharth Uppal, Jaydeep Chakraborty, and Srividya K. Bansal. 2021. Template-based Question Answering analysis on the LC-QuAD2.0 Dataset. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*. 443–448. https://doi.org/10.1109/ICSC50631.2021.00079

[4] Eleftherios Dimitrakis, Konstantinos Sgontzos, and Yannis Tzitzikas. 2020. A survey on question answering systems over linked data and documents. *Journal of intelligent information systems* 55 (2020), 233–259.

[5] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *International Semantic Web Conference*. Springer, 69–78.

[6] Paolo Ferragina and Ugo Scaiella. 2010. TAGME: On-the-Fly Annotation of Short Text Fragments (by Wikipedia Entities). In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management* (Toronto, ON, Canada) *(CIKM '10)*. Association for Computing Machinery, New York, NY, USA, 1625–1628. https://doi.org/10.1145/1871437.1871689

[7] Jorão Gomes, Rômulo Chrispim de Mello, Victor Ströele, and Jairo Francisco de Souza. 2022. A Hereditary Attentive Template-based Approach for Complex Knowledge Base Question Answering Systems. *Expert Systems with Applications* 205 (2022), 117725. https://doi.org/10.1016/j.eswa.2022.117725

[8] Jorão Gomes Jr, Rômulo Chrispim de Mello, Victor Ströele, and Jairo Francisco de Souza. 2022. A study of approaches to answering complex questions over knowledge bases. *Knowledge and Information Systems* 64, 11 (2022), 2849–2881.

[9] Jorão Gomes Jr., Rômulo Chrispim de Mello, Victor Ströele, and Jairo Francisco de Souza. 2021. LC-QuAD 2.1. https://doi.org/10.5281/zenodo.5508297

[10] Xixin Hu, Xuan Wu, Yiheng Shu, and Yuzhong Qu. 2022. Logical Form Generation via Multi-task Learning for Complex Question Answering over Knowledge Bases. In *Proceedings of the 29th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, Gyeongju, Republic of Korea, 1687–1696. https://aclanthology.org/2022.coling-1.145

[11] Heewon Jang, Yeongtaek Oh, Seunghee Jin, Haemin Jung, Hyesoo Kong, Dokyung Lee, Dongkyu Jeon, and Wooju Kim. 2017. KBQA: Constructing Structured Query Graph from Keyword Query for Semantic Search. In *Proceedings of the International Conference on Electronic Commerce* (Pangyo, Seongnam, Republic of Korea) *(ICEC '17)*. Association for Computing Machinery, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/3154943.3154955

[12] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. A Survey on Complex Knowledge Base Question Answering: Methods, Challenges and Solutions. *CoRR* abs/2105.11644 (2021). arXiv:2105.11644 https://arxiv.org/abs/2105.11644

[13] Nandana Mihindukulasooriya, Gaetano Rossiello, Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Mo Yu, Alfio Gliozzo, Salim Roukos, and Alexander G. Gray. 2020. Leveraging Semantic Parsing for Relation Linking over Knowledge Bases. *CoRR* abs/2009.07726 (2020). arXiv:2009.07726 https://arxiv.org/abs/2009.07726

[14] Saeedeh Momtazi and Zahra Abbasiantaeb. 2022. *Question Answering over Text and Knowledge Base*. Springer Nature.

[15] Sumit Neelam, Udit Sharma, Hima Karanam, Shajith Ikbal, Pavan Kapanipathi, Ibrahim Abdelaziz, Nandana Mihindukulasooriya, Young-Suk Lee, Santosh Srivastava, Cezar Pendus, et al. 2022. A benchmark for generalizable and interpretable temporal question answering over knowledge bases. *arXiv preprint arXiv:2201.05793* (2022).

[16] Ngonga Ngomo. 2018. 9th challenge on question answering over linked data (QALD-9). *language* 7, 1 (2018), 58–64.

[17] Kechen Qin, Yu Wang, Cheng Li, Kalpa Gunaratna, Hongxia Jin, Virgil Pavlu, and Javed A. Aslam. 2020. A Complex KBQA System using Multiple Reasoning Paths. *CoRR* abs/2005.10970 (2020). arXiv:2005.10970 https://arxiv.org/abs/2005.10970

[18] Gaetano Rossiello, Nandana Mihindukulasooriya, Ibrahim Abdelaziz, Mihaela Bornea, Alfio Gliozzo, Tahira Naseem, and Pavan Kapanipathi. 2021. Generative Relation Linking for Question Answering over Knowledge Bases. In *The Semantic Web – ISWC 2021*, Andreas Hotho, Eva Blomqvist, Stefan Dietze, Achille Fokoue, Ying Ding, Payam Barnaghi, Armin Haller, Mauro Dragoni, and Harith Alani (Eds.). Springer International Publishing, Cham, 321–337.

[19] Ahmad Sakor, Kuldeep Singh, Anery Patel, and Maria-Esther Vidal. 2019. FALCON 2.0: An Entity and Relation Linking Tool over Wikidata. *CoRR* abs/1912.11270 (2019). arXiv:1912.11270 http://arxiv.org/abs/1912.11270

[20] Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. 2023. Evaluation of ChatGPT as a Question Answering System for Answering Complex Questions. *arXiv preprint arXiv:2303.07992* (2023).

[21] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*. Springer, Springer International Publishing, Cham, 210–218.

[22] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 2017. 7th open challenge on question answering over linked data (QALD-7). In *Semantic Web Challenges: 4th SemWebEval Challenge at ESWC 2017, Portoroz, Slovenia, May 28-June 1, 2017, Revised Selected Papers*. Springer, 59–69.

[23] Zhiwen Xie, Zhao Zeng, Guangyou Zhou, and Tingting He. 2016. Knowledge base question answering based on deep learning models. In *Natural Language Understanding and Intelligent Applications: 5th CCF Conference on Natural Language Processing and Chinese Computing, NLPCC 2016, and 24th International Conference on Computer Processing of Oriental Languages, ICCPOL 2016, Kunming, China, December 2–6, 2016, Proceedings 24*. Springer, 300–311.

[24] Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2021. Rng-kbqa: Generation augmented iterative ranking for knowledge base question answering. *arXiv preprint arXiv:2109.08678* (2021).