Journal on Interactive Systems, 2025, 16:1, doi: 10.5753/jis.2025.5428

A hereditary attentive question answering framework for knowledge bases

Rômulo C. de Mello [**Federal University of Juiz de Fora** | ***romulo.mello@estudante.ufjf.br***]
Jorão Gomes Jr. [**Vienna University of Economics and Business** | ***jorao.gomes.junior@wu.ac.at***]
Jairo Francisco de Souza [**Federal University of Juiz de Fora** | ***jairo.souza@ufjf.br***]
Victor Ströele [**Federal University of Juiz de Fora**| ***victor.stroele@ufjf.br***]

Campus Universitário, Rua José Lourenço Kelmer, s/n - São Pedro, Juiz de Fora - MG, 36036-900

Received: 15 January 2025 • **Accepted:** 01 August 2025 • **Published:** 22 August 2025

Abstract: *Background.* The rapid growth of online data has made retrieving relevant information a challenging task, prompting the rise of Knowledge Base Question Answering (KBQA) systems that handle complex, multi-hop queries. *Purpose.* This extended work refines our previous pipeline by introducing structured dummy templates, a Hereditary Tree-LSTM (HTL) for classification, and more comprehensive analyses of entity recognition, property extraction, and SPARQL assembly. *Methods.* We enhanced the LC-QUAD 2.1 dataset with standardized templates and evaluated a flexible pipeline that integrates DeepPavlov, Falcon, SpaCy, qualifiers constraints, and reverse lookups. *Results.* Our experiments reveal that multi-tool entity recognition outperforms single-tool methods, while property extraction benefits from extended property sets and refined ranking strategies. Overall SPARQL correctness reaches up to 70–80% in mid-complex queries but remains lower in domain-specific subsets. *Conclusion.* The proposed synergy of NLP tools and refined dummy templates increases coverage for complex KBQA, though further improvements in morphological handling and specialized embeddings may be needed to address challenging multi-hop or niche queries comprehensively.

Keywords: KBQA, C-KBQA, Entity Recognition, Property Extraction, LC-QUAD

1 Introduction

The large volume of data available on the internet has made the task of finding relevant information even more challenging [Jang *et al*., 2017]. In this context, new information retrieval methods have enabled more intelligent searches, taking into account the context of the search. Additionally, advances in Natural Language Processing (NLP) research have allowed a better understanding of the search context, enabling Knowledge-based Question Answering (KBQA) frameworks to emerge as an efficient solution to meet this demand [Lan *et al*., 2021].

In particular, we focus on Complex Knowledge-Based Question Answering (C-KBQA), a subclass of KBQA that aims to answer questions involving multiple entities, properties, and reasoning steps such as comparisons, temporal relations, or conditions. This distinction allows for a more precise evaluation of systems capable of handling intricate question structures.

However, the complexity of the questions users ask remains a significant challenge for KBQA frameworks [Qin *et al*., 2020]. Questions may involve multiple concepts, conditional or comparative reasoning, and temporal aspects, requiring the system to understand both the mentioned entities and the relationships among them. For instance, questions such as "Which films directed by Quentin Tarantino were nominated for an Oscar?" or "Is the Eiffel Tower taller than the Statue of Liberty?" demand a deep understanding of entities, properties, and comparisons [Xie *et al*., 2016]. This complexity increases significantly when implicit context needs to be inferred, as in questions like "What is the country's capital?" [Hu *et al*., 2022].

Large language models (LLMs), such as GPT, have shown significant advances as Question-Answering (QA) systems, providing impressive results when answering natural language questions [Daull *et al*., 2023]. However, these models present limitations, especially in accessing information not included in their training data [Tan *et al*., 2023]. On the other hand, KBQA frameworks are designed to access structured knowledge bases, such as knowledge graphs, enabling more efficient retrieval of updated and domain-specific information. Previous work has explored the use of semantic knowledge graphs for integrating corporate data, demonstrating the potential of these approaches in precise information retrieval [Rolim *et al*., 2021].

Additionally, frameworks that evaluate the connectivity of entity pairs in knowledge bases have highlighted the importance of understanding relationships among entities to construct accurate answers [Jiménez *et al*., 2021b] and [Jiménez *et al*., 2021a]. Such approaches emphasize the relevance of robust entity and property extraction systems, as used in KBQA frameworks. Moreover, methods that evaluate entity similarity in RDF knowledge bases have advanced the semantic understanding of queries, improving the retrieval of contextually relevant data [Jiménez *et al*., 2022].

In this study, we propose a KBQA framework featuring a practical and flexible pipeline designed to handle complex knowledge-based questions. The pipeline is modularly structured, allowing for the replacement of entity recognition and property extraction models, as well as refinements in SPARQL queries using qualifier constraints. The flexibility of the proposed approach is inspired by studies demonstrat-

ing the effectiveness of customizable frameworks in different domains [Gomes Jr *et al*., 2022]. The proposed framework aims not only to improve the accuracy of responses but also to facilitate future adaptations to other languages and domains.

This paper is an extended and revised version of a previously published work [Mello *et al*., 2024]. We discuss the flexibility of KBQA frameworks and present new experiments to evaluate the application of the proposed techniques in more complex scenarios. The study includes enhancements in SPARQL query generation, with improved precision in entity and property recognition, as well as the integration of refinements in model adaptation. Our goal is to provide a practical and modular approach that can be tailored to different contexts and requirements, broadening the applicability of knowledge-based question-answering 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, especially when addressing complex questions. Complex questions typically involve multiple entities, multi-hop reasoning, numerical comparisons, temporal constraints, or label-based textual queries. These questions are inherently challenging, as evidenced by performance metrics reported in the literature—systems often struggle to achieve high accuracy consistently across diverse datasets. Consequently, advancements in KBQA frameworks frequently focus on incrementally improving the ability to handle complex structures and nuanced semantic relationships.

This section reviews significant contributions in four main areas: entity recognition, relation extraction, template matching, and frameworks.

2.1 Entity Recognition

Entity recognition is a fundamental task in KBQA, involving identifying entities mentioned in the user's query. One notable system in this area is *TagMe* [Ferragina and Scaiella, 2010], which introduced a technique for annotating short text fragments with relevant Wikipedia hyperlinks. Although developed in 2010, TagMe is still frequently referenced in the literature due to its efficiency in processing short and noisy texts.

The system uses a three-step process: spot identification, sense disambiguation, and annotation. A spot refers to a text fragment—such as a word or phrase—that may correspond to a Wikipedia entity. First, candidate spots are identified in the input text. Then, the system performs disambiguation by selecting the most appropriate Wikipedia page for each spot based on contextual and statistical information. Finally, the annotation step links each spot to its corresponding Wikipedia article, providing users with enriched context and facilitating access to background knowledge.

Another significant contribution in this field is Falcon [Sakor *et al*., 2019], 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 [Ngomo, 2018] and LC-QuAD 2.0 [Dubey *et al*., 2019].

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 [Mihindukulasooriya *et al*., 2020] 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 [Usbeck *et al*., 2017], QALD-9 [Ngomo, 2018], and LC-QuAD 1.0 [Trivedi *et al*., 2017].

Another innovative approach in relation extraction is presented by [Rossiello *et al*., 2021], 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 [Dileep *et al*., 2021], 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 [Gomes *et al*., 2022], 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 [Ye *et al*., 2021], 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 [Dutt *et al*., 2023] and WEBQSP [Sorokin and Gurevych, 2018] 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 [Chen *et al*., 2021], 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 Methods

This section provides a more detailed view of the study's methodological underpinnings, describing the choice and preparation of datasets, as well as the processes of template grouping and the creation of **dummy templates** with placeholders. In this expanded version, the objective is not only to describe *what* was done but also to explain *why* each step was essential for handling complex questions in a Knowledge-

Based Question Answering (KBQA) environment.

3.1 Methodological Introduction

The primary motivation for our C-KBQA framework is to ensure that complex questions—those involving multiple entities, temporal filtering, conditional counting, and chained relationships—can be consistently converted into coherent SPARQL queries. Such questions often require more than simply identifying one entity; they frequently include elements such as "What is the birth date of the director of a certain film, as long as the film received an award after 2000?" or "Which players were transferred to a different team before a specific year?" These examples reveal the variety of multi-hop operations and precise filters that must be managed. With this in mind, we looked for a dataset that both captured these scenarios of high complexity and allowed for fair performance assessment.

We focused on the *Large-scale Complex Question Answering Dataset (LC-QUAD) 2.0*, subsequently refined into *LC-QUAD 2.1*. In the following sections, we outline the key characteristics of each dataset, emphasizing the inconsistencies that led to LC-QUAD 2.1 and explaining how we organize and structure SPARQL templates for such complexity. Although the first subsection addresses why these data were chosen, subsequent subsections explain the semantic grouping of templates and the introduction of **dummy templates**—placeholders that simplify the mapping from questions to SPARQL queries.

3.2 LC-QUAD 2.0

LC-QUAD 2.0 [Dubey *et al*., 2019] was created to push beyond the limitations of earlier QA datasets by significantly expanding both the number of queries and the breadth of question types. Unlike prior resources that often featured only a few thousand simple questions, LC-QUAD 2.0 contains over 30,000 English-language questions addressing complex, multi-hop structures and advanced SPARQL features such as numeric and textual filters, `COUNT` operations, and temporal qualifiers. This magnitude of content was crowdsourced via the Amazon Mechanical Turk platform, ensuring broader linguistic variety and the inclusion of paraphrased versions for many questions.

A central goal in developing LC-QUAD 2.0 was to promote more realistic testing of how QA systems handle multi-faceted queries. Rather than restricting questions to a single triple pattern or straightforward relationships, the authors introduced scenarios requiring two or more interconnected facts, time-based conditions (for instance, an event that must occur before or after a given year), and textual constraints (involving `contains` or `STRSTARTS`). These additions compel systems to integrate multiple knowledge graph edges, interpret qualifiers (such as an event's date or location), and cope with more diverse language variations. To facilitate this diversity, the dataset was constructed around both Wikidata and DBpedia 2018, allowing queries to tap into distinct but overlapping knowledge graphs and highlight potential differences in how systems map question text to each resource's ontology.

Another distinctive contribution is the deliberate inclusion of paraphrased questions. The dataset's creators recognized that QA systems trained exclusively on a single wording of a query might overfit to specific linguistic cues. By asking Mechanical Turk workers to restate or paraphrase each question, LC-QUAD 2.0 ensures that machine-learning approaches cannot rely solely on narrow syntactic patterns. This requirement to handle varying formulations of the same intent is especially relevant for advanced queries where multi-hop reasoning or qualifiers are combined. Systems that succeed on LC-QUAD 2.0, therefore, must prove robust against rephrasings and synonyms.

Although LC-QUAD 2.0 represented a major leap in complexity and quantity over its predecessor LC-QUAD 1.0, certain practical issues emerged. The large-scale crowdsourcing methodology led to occasional inconsistencies between question text and the assigned SPARQL query, along with instances of overlap or duplication in the data. Some queries proved too short or vague to give the system meaningful clues, while others veered toward extreme length or unwieldy lexical constructions. These challenges made it harder to ascertain whether errors arose from a model's limitations or from mismatched or duplicated entries. Nonetheless, the essential complexity the dataset introduced—covering two-intent questions, qualifier usage, date-based restrictions, and textual substring operations—marked a turning point in the field. No earlier dataset had offered such an extensive set of natural language queries with corresponding SPARQL templates across DBpedia and Wikidata.

Researchers seeking to build or evaluate complex QA systems found multiple uses for LC-QUAD 2.0. One common application involved entity-linking or predicate-linking tasks, where the dataset's short, sometimes paraphrased questions forced algorithms to handle ambiguous references and partial textual matches. Others employed LC-QUAD 2.0 for training end-to-end question-answering pipelines that generate SPARQL queries. The variety of advanced operations present in the dataset—a reflection of the depth of Wikidata's and DBpedia's schemas—stimulated the development of machine-learning approaches that handle multi-hop patterns and complicated string filters more gracefully.

3.3 LC-QUAD 2.1

LC-QUAD 2.1 [Gomes Jr. *et al*., 2021] emerged as a refined and standardized update to the large-scale LC-QUAD 2.0 dataset. Building upon the original collection of more than 30,000 questions, its creators undertook a comprehensive review to eliminate duplicate entries, remove malformed or excessively short items, and address inconsistencies between question text and the associated SPARQL queries [Gomes *et al*., 2022]. The resulting dataset preserves the core complexity of multi-hop reasoning, numeric and textual filters, and advanced operators, yet offers a more uniform set of mappings from natural language to formal query structures. By mitigating issues like duplicated questions and mismatched references, LC-QUAD 2.1 reduces the possibility of QA systems failing for reasons unrelated to genuine semantic or lexical challenges.

A defining enhancement introduced in LC-QUAD 2.1 was the concept of "dummy" elements within each SPARQL query. Rather than embedding fixed QIDs or numeric values in the query, placeholders such as `DUMMY_S` (for subjects), `DUMMY_P` (for properties), `DUMMY_O` (for objects), and `DUMMY_F` (for numeric or string filters) replaced the original references. This strategy separates the question's broad logical shape—for example, the number of triple patterns or the presence of filters—from its specific data-level details. Consequently, the pipeline can be split into two distinct phases. In the first phase, the system classifies a user query to determine which structural "template" it matches. In the second, the recognized placeholders are filled with the relevant entity IDs, property IDs, or numeric/string constraints discovered by entity- and property-linking modules.

This two-phase design streamlines experimentation in several ways. It explicitly distinguishes structural misinterpretation (choosing the wrong SPARQL "shape") from errors in entity linking or property mapping (supplying the wrong QIDs or thresholds). If the pipeline is already confident in the structural template, any subsequent errors become easier to localize and correct, often without having to retrain the entire model. Moreover, dummy placeholders allow incremental improvements in entity recognition or property detection to be introduced without disrupting the dataset's underlying template library. As a result, a flawed final SPARQL no longer conflates poor structural analysis with simple linking missteps, making system failures more traceable to specific components. This modularity in query building has proven particularly valuable when researchers experiment with new entity-linking strategies or domain-specific expansions.

By isolating each question's logical skeleton and its data-level references, LC-QUAD 2.1 presents a cleaner testbed for KBQA systems. Whether a query demands two or three hops, textual substring checks, or date-qualifier logic, the dummy format ensures the system's classification step accurately pins down which query form applies, while the linking step concentrates on populating placeholders with the correct QIDs or numeric constraints. In this sense, LC-QUAD 2.1 retains the multifaceted complexity of LC-QUAD 2.0 but provides a more stable platform for diagnosing the root cause of system errors and advancing the field of complex KBQA.

3.3.1 Limitations and Threats to Validity

LC-QUAD 2.0 was created through large-scale crowdsourcing, a process that produced an impressively diverse benchmark but also introduced substantial noise: near-duplicate questions, ultra-short prompts that resemble keyword searches, malformed characters, and mismatches between the question text and the corresponding SPARQL query. LC-QUAD 2.1 addresses these issues through a cleaning phase, but the very act of filtering the data introduces its own set of threats to validity.

The review relies on a few generic heuristics—length and character checks, duplicate detection, and consistency tests that check for the existence of referenced entities and properties. While these rules were chosen for their simplicity and transparency, they inevitably reflect subjective design choices. Different thresholds for what is considered "too short" or alternative strategies for identifying duplicates may result in a slightly different corpus. Similarly, edge cases may go unnoticed or be inadvertently removed, leaving residual noise or eliminating legitimate linguistic variety.

A limitation arises from the evaluation protocol. Performance is assessed by the exact string match between a predicted SPARQL query and the gold standard. A semantically correct query that follows an alternative but valid property path is marked as erroneous, while a syntactically perfect query may still produce an erroneous answer if the knowledge graph has evolved. Relying on this metric alone risks under- or overestimating practical effectiveness; further evaluations based on answer sets or user-centric studies would provide a more complete picture.

Finally, entity and property linking depends on specific versions of external NLP libraries and a specific snapshot of Wikidata. Updates to these tools or the underlying graph may change tokenization thresholds, embedding spaces, and feature frequencies, meaning that future replications may produce different results. Periodic reevaluation with updated tools, alternative datasets, and broader metrics remains essential to monitor genuine progress on KBQA.

3.4 Creating Dummy Templates

A central refinement introduced by LC-QUAD 2.1, in comparison to LC-QUAD 2.0, was the decision to separate the fixed logical form of each SPARQL query from the specific QIDs, property labels, and numeric or textual filters that appeared in it. In LC-QUAD 2.0, each query was fully specified and directly referenced entities such as `wd:Q3772` (Quentin Tarantino) and properties like `wdt:P57` (directed by). However, for handling complex queries in a more flexible, generalizable fashion, LC-QUAD 2.1 introduced placeholders in place of these explicit identifiers. This step is referred to as "dummy-fication."

During dummy-fication, each original SPARQL query was rewritten with placeholders (`DUMMY_S` for subjects, `DUMMY_P` for predicates, `DUMMY_O` for objects, and `DUMMY_F` for numeric or string filters). For example, a fully specified query involving two properties and a numeric cutoff would be converted to a structure that indicates the same number of triple patterns and filters, but no longer binds them to a specific QID or integer. Through this process, the dataset preserves only the "shape" of each query (such as whether it is `ASK` or `SELECT`, how many hops it contains, or how many comparison filters are included) without tying those shapes to fixed IDs or values.

Performing this dummy creation before any classification or grouping steps has practical benefits. It clarifies that two queries referencing different QIDs but using the same underlying pattern (for instance, "Which films directed by X were nominated for Y after year Z?") share a single structural form. Consequently, the pipeline can focus on the logic of the query—how many relationships or filters it contains—rather than on the specific details of each entity or property. Table 1 illustrates a few dummy templates, highlighting how the "raw" references get replaced by placeholders while preserving the SPARQL skeleton.

By ensuring every query adheres to this placeholder-based

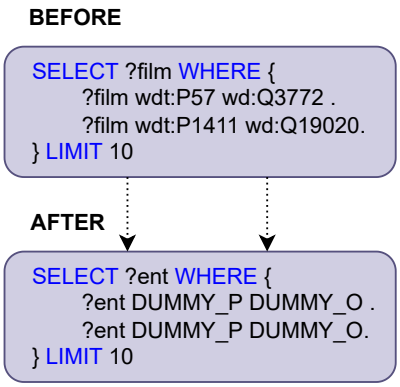

Figure 1. Illustration of how raw SPARQL becomes a dummy template with placeholders

format, the dataset becomes more amenable to subsequent pipeline tasks such as grouping queries by complexity or training a model that classifies questions according to their structure.

3.5 Grouping Templates by Semantic Proximity

After converting all LC-QUAD 2.0 SPARQL queries into *dummy templates* (i.e., replacing explicit entity IDs, property IDs, and numeric/string values with placeholders), the next essential step in LC-QUAD 2.1 was to cluster these templates according to their structural and semantic features. This process yielded a total of **13 groups** (indexed 0 through 12), covering **29 distinct dummy templates**. Each group comprises one or more templates that share a common logic or level of complexity, such as whether the query is `ASK` or `SELECT`, if it requires multi-hop reasoning, or whether it applies numeric/string filters. Table 1 summarizes the groups, noting the number of templates in each, a brief description, and a single *Dummy SPARQL example* from that group.

By grouping dummy templates that share common structures, LC-QUAD 2.1 provides a concise map of how query complexity progresses. A single `ASK` plus numeric filter might belong to Group 0, a two-hop `SELECT` might reside in Group 4 or 5, and textual label checks fall under Groups 7, 9, or 11. This organization allows the pipeline to narrow down the relevant set of template "shapes" when faced with a user question that, for instance, demands multi-hop plus a textual filter.

Although the table above highlights the 13 main groups, each group can contain multiple related templates, reflecting minor variations in operators, filters, or multi-hop steps. For instance, Group 0 encloses 4 templates (IDs 0.1–0.4), whereas Group 4 includes 5 (4.1–4.5). This arrangement ensures that if new SPARQL forms arise—e.g., a specialized subquery or advanced date operator—one can extend or refine an existing group or define a new one, keeping the classification consistent.

Clustering these dummy templates by semantic proximity underpins the entire pipeline's ability to locate the correct SPARQL skeleton quickly. Once a user query is recognized as, say, a multi-hop `SELECT` with date-based filtering, the system knows exactly which group (and subsequently which specific template) to match. This strategy significantly reduces the complexity of forming valid SPARQL queries, since the pipeline only needs to fill placeholders (`DUMMY_S`, `DUMMY_P`, `DUMMY_O`, `DUMMY_F`) with the correct QIDs or numeric thresholds after the structure is identified.

Moreover, the defined dummy templates explicitly separate the structural logic of a query from the specific entities and properties involved. This structural abstraction facilitates the reuse or adaptation of templates across different datasets, provided the questions share similar semantic or logical patterns. For example, questions from alternative datasets such as QALD [Perevalov *et al*., 2022] or KQA Pro [Cao *et al*., 2022], which also involve multi-hop reasoning, numeric filtering, temporal constraints, or label-based textual queries, can often be mapped onto existing structural groups identified in this study.

To apply the templates to a new dataset, it may be necessary to first verify whether the dataset's query structures are already represented in the existing Hereditary Tree-LSTM (HTL) templates. If a new query structure emerges, it must be incorporated into the HTL templates. Subsequently, a fine-tuning process is essential, either by integrating the new template within an existing semantic group or creating a new group specifically tailored to accommodate this additional structure.

The hierarchical organization into structural groups based on complexity and semantic characteristics ensures scalability and systematic growth. This allows the framework to adapt effectively as new queries with minor structural variations appear, integrating these seamlessly as extensions or refinements within existing groups.

Nonetheless, while structural templates form a solid basis for generalization, their successful application largely depends on the adaptability of the entity and property recognition modules. Therefore, generalizing to new datasets necessitates the incorporation of domain-specific embeddings, refined property ranking strategies, and specialized morphological processing tailored to the terminology and characteristics of the target knowledge base. These targeted enhancements are critical to maintaining high pipeline accuracy, even when processing novel or domain-specific query formulations.

3.6 Hereditary Tree-LSTM (HTL)

After defining the overall strategy for grouping SPARQL templates and refining the dataset into dummy structures, a crucial question arises: how can the system determine which template best matches a user's complex question, especially when multiple relational hops or filters may be involved? A purely rule-based or semantic-parsing approach might try to construct logical forms from scratch, which can be computationally expensive and prone to errors if the user's query contains multiple constraints or unusual syntactic cues. To address this challenge, we introduced the **Hereditary Tree-LSTM (HTL)**, drawing on the concepts put forth in [Gomes *et al*., 2022], as an efficient and robust means of classifying

Table 1. All 13 groups (0–12) of dummy templates in LC-QUAD 2.1, with representative examples

Group ID	#Templates	Description	Dummy SPARQL Example
0	4	Basic ASK queries with at most one numeric filter (yes/no checks)	`0.2: ASK WHERE { DUMMY_S DUMMY_P ?obj FILTER(?obj = DUMMY_F) }`
1	1	ASK queries referencing two relationships	`1.0: ASK WHERE { DUMMY_S DUMMY_P DUMMY_O . DUMMY_S DUMMY_P DUMMY_O }`
2	2	Single-hop SELECT retrieval, typically one relationship	`2.2: SELECT DISTINCT ?answer WHERE { DUMMY_S DUMMY_P ?answer }`
3	2	"How many..." questions using COUNT (single-hop)	`3.1: SELECT (COUNT(?sub) AS ?value) { ?sub DUMMY_P DUMMY_O }`
4	5	Multi-hop SELECT, often 2 steps with an intermediate variable	`4.2: SELECT ?answer WHERE { DUMMY_S DUMMY_P ?answer . ?answer DUMMY_P DUMMY_O }`
5	2	Multi-hop SELECT with extra reference to the same subject/object	`5.1: SELECT ?obj WHERE { DUMMY_S DUMMY_P ?s . ?s DUMMY_P ?obj . ?s DUMMY_P DUMMY_O }`
6	2	Multi-hop plus ORDER BY and LIMIT	`6.1: SELECT ?ent WHERE { ?ent DUMMY_P DUMMY_O . ?ent DUMMY_P ?obj . ?ent DUMMY_P DUMMY_O } ORDER BY ASC(?obj) LIMIT 10`
7	4	SELECT queries with string/date-based filters	`7.2: SELECT ?value WHERE { DUMMY_S DUMMY_P ?s . ?s DUMMY_P ?x FILTER(contains(?x,'DUMMY_F')) . ?s DUMMY_P ?value }`
8	1	SELECT retrieving multiple answers in one step	`8.0: SELECT ?ans_1 ?ans_2 WHERE { DUMMY_S DUMMY_P ?ans_1 . DUMMY_S DUMMY_P ?ans_2 }`
9	2	Label-based constraints, partial string checks, lang(...)	`9.2: SELECT DISTINCT ?sbj ?sbj_label WHERE { ?sbj DUMMY_P DUMMY_O . ?sbj DUMMY_P ?sbj_label . FILTER(STRSTARTS(lcase(?sbj_label),'DUMMY_F')) . FILTER(lang(?sbj_label)='DUMMY_F') } LIMIT 10`
10	1	SELECT retrieving multiple columns (?value1, ?obj) in one hop	`10.0: SELECT ?value1 ?obj WHERE { DUMMY_S DUMMY_P ?s . ?s DUMMY_P ?obj . ?s DUMMY_P ?value1 }`
11	2	Repeated references plus textual filters for labels	`11.2: SELECT DISTINCT ?sbj ?sbj_label WHERE { ?sbj DUMMY_P DUMMY_O . ?sbj DUMMY_P DUMMY_O . ?sbj DUMMY_P ?sbj_label . FILTER(STRSTARTS(lcase(?sbj_label),'DUMMY_F')) . FILTER(lang(?sbj_label)='DUMMY_F') } LIMIT 10`
12	1	Multi-value SELECT retrieving two results from the same intermediate node	`12.0: SELECT ?value1 ?value2 WHERE { DUMMY_S DUMMY_P ?s . ?s DUMMY_P DUMMY_O . ?s DUMMY_P ?value1 . ?s DUMMY_P ?value2 }`

complex natural language questions into their corresponding dummy templates.

The selection of the HTL is grounded in prior work that provides a direct comparative analysis against baseline models, effectively serving as an ablation study for this component of our pipeline. The study by [Gomes *et al*., 2022], which introduced the HTL, evaluated it on the LC-QUAD 2.1 dataset and demonstrated its advantages. Firstly, when compared against traditional machine learning approaches for the same task, such as the XGBoost classifier proposed by [Dileep *et al*., 2021], the HTL achieved a higher accuracy of 73.3% versus 67.2%. Secondly, an internal ablation within the study confirmed the specific contribution of the "hereditary attention" mechanism; the full HTL model outperformed a standard Tree-LSTM version without this mechanism by achieving an accuracy of 73.3% compared to 72.9%. Based on this empirical evidence, the HTL was chosen as the most suitable classifier for our framework.

A standard Tree-LSTM departs from the linear token-processing found in traditional LSTM architectures by operating directly on a syntactic tree, such as one produced by a dependency parser. Each node in this tree represents a token, while edges reflect syntactic or semantic relationships between words, like subject-verb or adjective-noun links. By summing the hidden states of child nodes, the Tree-LSTM captures the hierarchical structure of language in a way that a linear model may overlook. This structural advantage becomes critical in questions that reference multiple entities, date thresholds, or chained comparisons, since relevant words are not always linearly adjacent in the sentence.

HTL extends this by incorporating what is called a "hereditary attention" mechanism. Rather than having a single attention layer that may overlook subtle interactions, each subtree computes local attention signals that highlight tokens

or phrases deemed most significant. These attention signals then pass upward in a bottom-up fashion. By the time the hidden representations accumulate at the root node of the tree, the model has integrated key features from lower-level phrases—references to comparative operators such as "greater than" or date markers like "after 2000" or relational patterns suggesting multi-hop logic. This richer, aggregated representation at the root node is then passed to a classification layer, typically a softmax, to decide which dummy template is structurally most suitable for the user's question.

The "hereditary" aspect of HTL is especially useful in questions where relevant pieces of information are scattered in different branches of the sentence. In a linear LSTM, it might be difficult to keep track of these scattered clues without heavily engineered gating or attention modules. By contrast, the Tree-LSTM natively deals with linguistic hierarchies, and the hereditary attention ensures that essential fragments are not lost as intermediate child nodes pass their states to the parent node. This approach proves especially powerful for complex queries that combine numeric or temporal conditions with multi-hop relationships, since each child node can measure local significance and transmit that forward.

In practical terms, the decision process with HTL involves two major phases. The first is the construction of the tree representation. The question is parsed using a syntactic or dependency parser, producing a tree whose nodes correspond to tokens. Each node is associated with an embedding (for instance, via pretrained word embeddings or contextual representations) and a dependency relation to its parent. The second phase is the bottom-up accumulation of hidden states and attentional weights. Each node computes, for each child, how relevant it is to the overarching meaning. These local attentions are aggregated, culminating in a top-level vector. A final classification stage maps this top-level vector to a dummy template ID.

Several advantages emerge from this design. One advantage is that HTL substantially reduces the need to enumerate all possible logical forms or to rely on purely text-based classification methods that might fail to appreciate the compositional nature of the sentence. Another advantage lies in providing better interpretability: by inspecting the attention distributions at each node, one can see which parts of the question the Tree-LSTM deems most crucial. This can be invaluable when explaining why a given template was chosen over alternatives. A potential limitation is that it requires reliable dependency parsing; errors in the parse tree can propagate upward and cause misclassification. Nevertheless, for well-formed sentences such as those curated in LC-QUAD 2.1, this risk is mitigated by the dataset's improved consistency.

Once the HTL selects the best matching template, the process of constructing the final SPARQL query is greatly simplified. The question no longer needs an entirely new logical form; it only requires filling the dummy slots (`DUMMY_S`, `DUMMY_P`, `DUMMY_O`, `DUMMY_F`) with the right entities, properties, or filter values. This separation between structure classification and slot filling yields both computational efficiency and clarity: the system pinpoints the structural skeleton first, then calls upon entity-extraction modules to fill in details. In the context of LC-QUAD 2.1, whose queries already come in dummy form, HTL stands out as a stable and powerful means of bridging the gap between the question's linguistic complexity and the final query's logical architecture.

The accurate classification provided by HTL greatly streamlines the KBQA pipeline by clearly defining the structural template to be used. However, the task remains incomplete without precisely identifying and linking the specific entities and properties within the user's query to the corresponding Wikidata identifiers (QIDs). This necessity underscores the critical role of robust entity, property, and filter extraction tools, which are introduced in detail in the following subsections.

3.7 Tools for Entity Extraction

Once the HTL indicates which dummy template structure should apply to a user's question, the pipeline still needs to identify the exact entities mentioned so it can fill placeholders (e.g., `DUMMY_S`, `DUMMY_O`) with the right QIDs. Although entity linking seems straightforward when questions reference well-known concepts such as "Quentin Tarantino" or "Oscar," it can be more difficult with ambiguous or lesser-known items, domain-specific abbreviations, and nonstandard phrasing. To address these diverse scenarios, we employ three tools in parallel: DeepPavlov [DeepPavlov Team, 2024], Falcon 2.0, and spaCy. Each tool focuses on detecting named entities and linking them to potential Wikidata identifiers, but they do so using different methodologies, thereby reducing the chance that one tool's blind spots will lead to missed or incorrect matches.

DeepPavlov. It provides a wide range of pretrained NLP models, including pipelines for both named entity recognition (NER) and entity linking. When a user query is processed by DeepPavlov, the system first segments the text to identify candidate entity boundaries. For instance, it can discern that "New York City" is a single entity mention, rather than erroneously splitting "New York" and "City." Once the candidate mentions are extracted, DeepPavlov's linking model uses embedding-based similarity to map each mention to potential Wikidata entries. This stage can struggle with short or rare mentions that do not exhibit strong embedding matches to any known label or alias in the knowledge base. However, DeepPavlov's flexibility in customizing models or training them on domain-specific text is advantageous if the pipeline needs to adapt to specialized vocabularies (e.g., medical or technical domains).

Falcon 2.0. Adopts a rule-driven approach to entity and relation linking, emphasizing tokenization, morphology, and N-Gram tiling. It is particularly adept at capturing short compound forms, multiword expressions, or property-like phrases that may not appear in standard NER outputs. For example, it can detect "operating income'' or "manager position'' as meaningful references to possible relations or specialized entity phrases. Once Falcon identifies these candidate tokens, it queries a local index of Wikidata labels and aliases and ranks the retrieved entities by textual similarity. Although Falcon 2.0 can sometimes over-prioritize close

string matches in ambiguous cases (e.g., multiple items sharing a term like "Ford"), its ability to identify less common or domain-specific phrases complements the embedding-based coverage gaps that DeepPavlov might leave.

spaCy. Offers a fast, comprehensive NLP pipeline that performs tokenization, part-of-speech (POS) tagging, dependency parsing, and built-in NER. We use spaCy not just for its NER component but also to parse each query into a structured dependency tree, which supports the HTL-based classification and helps interpret multiword sequences. spaCy's default entity linker, while somewhat generic, often succeeds on widely used entity names or standard lexical items. Its speed and practical approach to common named entities make it a useful baseline for everyday queries, and its token and syntactic boundaries allow the other two tools to coordinate over consistent phrase segments. In many cases, spaCy can rapidly resolve simpler queries or confirm well-known entities so that the pipeline can focus more time on ambiguous or domain-specific references.

In practice, all three tools run concurrently on each user query, producing separate lists of entity mentions and their top candidate Wikidata links. We then merge these lists using a set of heuristic rules that check for overlaps or disagreements. If two or more tools converge on the same entity mention, confidence in that match increases significantly. Conversely, if DeepPavlov proposes a mention that Falcon 2.0 and spaCy do not see (or strongly disagree with), the pipeline may attempt partial substring matching or check the local neighborhood of tokens to see if the mention is plausible. In ambiguous cases, we also consider how frequently a proposed entity ID appears in Wikidata and whether it aligns with the query's overall domain or syntactic clues from spaCy's dependency parse. This ensures that a single tool's shortcoming—like missing a name variant or failing to parse compound terms—does not cause the pipeline to discard a valid candidate.

A recurring challenge arises when user questions involve abbreviations (e.g., "NYC" instead of "New York City"), short acronyms, or domain-specific synonyms rarely seen in ordinary embeddings. Such references often result in low matching scores for embedding-based approaches like DeepPavlov. Falcon 2.0's rule-based expansions can partially help, but only if the relevant aliases appear in its index. spaCy may detect the tokens but lack a strong domain linker. In these cases, the pipeline checks morphological expansions, or potential substring matches across all candidate sets.

Advantages of a Multi-Tool Strategy. By combining spaCy, DeepPavlov, and Falcon 2.0, the pipeline loads two local models (spaCy and DeepPavlov) and issues a lightweight HTTP request to a Falcon service. The chief overhead therefore lies in memory rather than in wall-clock time. spaCy alone keeps a compact statistical model resident; incorporating DeepPavlov brings transformer weights into memory and typically pushes the resident-set size into the multi-gigabyte range on a standard workstation. Falcon, by contrast, adds virtually no local memory cost because its computation occurs on an external service and the client stores only a small request buffer.

Once the additional models are loaded, per-question latency rises only modestly. In informal timing on commodity CPUs, total entity-linking time remains comfortably below the sub-second threshold that is generally considered interactive for QA applications; the HTTP round-trip to Falcon contributes just a small fraction of that delay. In other words, the pipeline trades extra RAM for a noticeably higher hit rate on tricky entity mentions without imposing a prohibitive runtime penalty.

Practically speaking, the choice boils down to hardware constraints. A machine with sufficient RAM can keep both local models in memory and still respond promptly, whereas a memory-constrained device might fall back to a lighter configuration or load models on demand. For complex KBQA, the reduced risk of missing critical entities usually justifies the additional memory footprint. Furthermore, the architecture remains modular: newer or domain-specific linkers can be swapped in with minimal code changes, preserving the same balance between robustness and latency.

3.8 Approach for Property Extraction

Whereas entity extraction resolves `DUMMY_S` (subject) and `DUMMY_O` (object) placeholders, another key question is how to find the properties—i.e., the `DUMMY_P` slots—that define the relationships or attributes mentioned in the user's query. Sometimes the query text is explicit (e.g., "*directed by*"), making detection relatively easy. More often, especially in multi-hop questions, relational clues may be implicit ("*starring in a film*" might map to "*cast member*" in Wikidata). Disambiguation is equally vital here, since Wikidata hosts thousands of properties with labels that may overlap or sound similar.

Initial Property Hints from Extraction Tools. Falcon 2.0 is often the most direct source for early property hints, since it is designed to locate both entities and relations in short text. When the user's question includes a phrase like "*nominated for an Oscar*", Falcon 2.0 might produce "wdt:P1411" (nominated for) as a candidate. DeepPavlov's main emphasis is on entity linking, though it can occasionally flag property-like tokens. spaCy itself does not inherently propose property IDs, but can help segment the question to see where relational phrases appear.

Difficulties in Multi-Hop or Implicit Relations. Challenges multiply if the question calls for more than one property (e.g., "*Which politician's spouse held office before 2000?*") or if the user's wording does not align neatly with Wikidata property labels. The HTL's template classification might reveal that the question structure involves two relationships plus a temporal filter, but it does not specify which properties to use. If the user's text yields only partial lexical clues, the pipeline may need to guess among multiple property candidates. This is where we incorporate both textual cues and knowledge-graph lookups to increase accuracy.

SPARQL Reverse Lookup for Coverage. If a recognized entity appears as a subject, we can query Wikidata for all properties where is the subject of a statement. We do the same if seems to be an object. This "reverse lookup" step is crucial in capturing less obvious properties not explicitly suggested by the user's phrasing or by initial text-based extraction. However, for prominent entities with hundreds of statements, retrieving so many property IDs can become overwhelming. Thus, we filter and rank them based on semantic similarity to the question text and any relevant contexts extracted by spaCy's dependency analysis.

Ranking Property Candidates. Because the reverse lookup or textual hints may produce multiple possibilities, we apply a final **ranking** step to each candidate property. We compare the property's label or description to key words in the question, using word-embedding vectors (for instance, from FastText). If the user's query references "*acted in movie*" a property whose label includes "*cast member*" might score well. Conversely, a property about "*producer*" or "*distributor*" might rank lower. Only the top few property candidates (commonly the top 5) progress to the final pipeline stage, preventing exponential growth in potential SPARQL combinations.

Qualifiers and Constraints. Even when the correct property is identified, Wikidata can embed qualifiers specifying contextual constraints, such as date ranges or roles. If the question implies these qualifiers (e.g., "*during the 1990s*"), ignoring them can yield an incomplete or incorrect final query. For relevant properties, the pipeline checks whether qualifiers exist that match the question's temporal or contextual conditions. Such qualifiers might be introduced as additional statements or filter clauses in the final query, ensuring alignment with the user's multi-layered request.

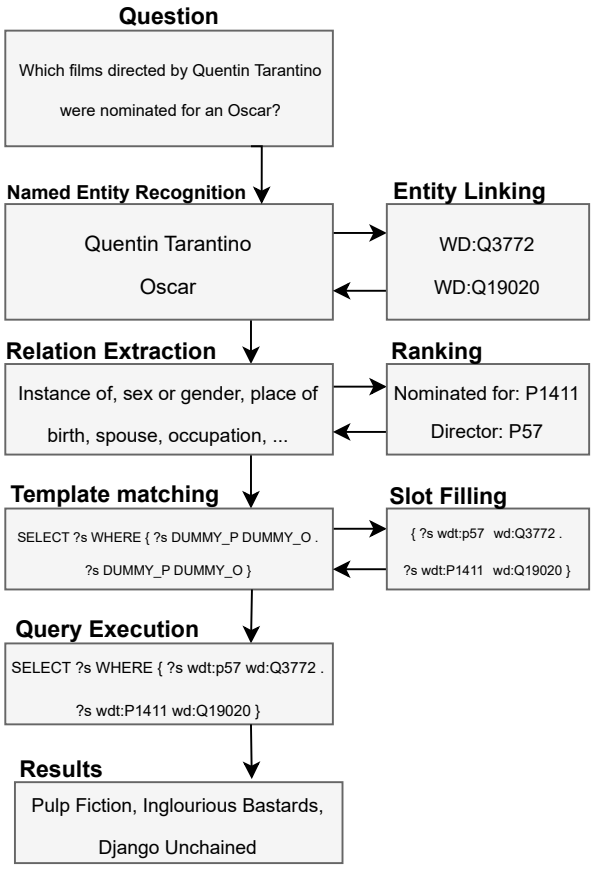

Figure 2. Fluxogram illustrating how the pipeline integrates template selection (HTL), entity recognition, property extraction, and final alignment with LC-QUAD 2.1's gold SPARQL.

3.9 Pipeline Integration and Remaining Steps

As depicted in Figure 2, by the time the pipeline has identified a suitable dummy template through the HTL and consolidated a set of entity and property matches from the multi-tool extraction and ranking modules, it can proceed to fill each placeholder (`DUMMY_S`, `DUMMY_P`, `DUMMY_O`, and optionally `DUMMY_F`) with the selected QIDs, property IDs, and any numeric or string constraints. When qualifier constraints have been detected, these too must be integrated into the final template structure to capture time-sensitive or contextual details.

Instead of executing this fully populated SPARQL query on a live endpoint, the system compares the filled template directly to the gold-standard SPARQL provided by LC-QUAD 2.1. This approach allows us to confirm whether the discovered entities, properties, and constraints align precisely with those in the question's reference query—an indication that the pipeline arrived at the correct combination of placeholders and slot values for that template. We measure performance at multiple stages (Correct Entities, Correct Properties, and Correct SPARQL) without relying on any actual query execution. A misplaced entity disrupts `DUMMY_S`, a property mismatch impacts `DUMMY_P`, and broader structural errors may point to shortcomings in the HTL-based template selection itself.

By leveraging the LC-QUAD 2.1 gold SPARQL queries, we do not need to issue real-time queries against a knowledge base to determine accuracy. Consistency with the reference indicates that the pipeline successfully reproduced the question's logic—covering entities, properties, filters, and qualifiers. Where mismatches arise, more targeted refinements can be made: for instance, repeated minor property errors may suggest improvements in the ranking module or reverse lookups; systematic placeholder mismatches might point to gaps in entity linking or qualifier integration.

This design not only streamlines our evaluation but also preserves a modular structure. Each subsystem—classification, entity extraction, property selection, or template refinement—can be updated or replaced, while the gold SPARQL references in LC-QUAD 2.1 remain a stable benchmark for correctness. As a result, we can continuously refine or expand the pipeline (through new extraction models, advanced ranking heuristics, or expanded dummy templates) while maintaining a consistent means of verifying overall alignment with the intended SPARQL.

4 Results and Discussion

This section provides a more detailed analysis of how each phase of our KBQA pipeline—entity recognition, property extraction, and final SPARQL query alignment—performs under multiple experimental setups. We tested configurations ranging from single-tool approaches (e.g., **DP**: DeepPavlov alone) to pairwise integrations (e.g., **DP+F**: DeepPavlov + Falcon), and finally a three-tool combination with additional refinements (qualifier constraints and reverse property lookups). In the following tables, the columns indicate the entity or property accuracy (percentage of queries where *all* relevant entities or properties were correctly identified) and the final SPARQL completeness (the percentage of queries that fully matched the gold-standard structure in LC-QUAD 2.1).

4.1 Overall Results from Multiple Combinations

We begin by noting that simpler pipelines using a single tool (**DP**, Falcon, or SpaCy) often handle the more elementary queries in LC-QUAD 2.1 (e.g., Groups 0, 1, or 2) reasonably well, as these queries typically reference only one or two entities without complex filters or multi-hop reasoning. However, once queries introduce multiple relationships or require unusual constraints (especially in Groups 4, 6, 9, and 11), single-tool approaches generally fall short. Pairwise integrations improve coverage, but the best results consistently emerge from the full three-tool setup (**DP+F+S+Q**).

Groups are identified in LC-QUAD 2.1 to represent different structural SPARQL patterns (see Section 3.5 for details). For instance, Group 0 includes basic `ASK` queries, while Group 4 covers two-hop `SELECT` queries, and Group 9 focuses on label-based textual constraints. When reading the tables below, it helps to remember that:

- Lower group IDs (0–3) usually represent simpler queries like `ASK` or single-hop `SELECT` with minimal filters.
- Mid-range group IDs (4–7) reflect multi-hop queries or more intricate operations, such as date-based filters.
- Higher group IDs (8–12) handle either multi-value retrieval, label-based textual searches, or repeated references, which can be trickier for standard extraction tools.

4.2 Entity Recognition in Each Group

Table 2 reports the entity recognition accuracy across 13 groups, comparing six different tool configurations. The metric indicates the proportion of queries in which *all* entities were correctly identified. If even one entity was missed or mislabeled, the entire query is considered incorrect at the entity level.

Key Observations. Groups 0, 1, 2, and 3 often reference single-hop or simpler relations, so entity coverage easily exceeds 60–70% in most multi-tool setups. In particular, Group 3 sees an especially large jump in accuracy when using more than one tool (e.g., from 44.30% in DP+F to 94.30% in F+S), which suggests that the counting queries often rely on subtle cues for identifying the subject entity. By contrast, Groups 6 and 9 stand out as notably difficult, with the triple configuration (**DP+F+S+Q**) only reaching about 42.62% (Group 6) and 53.96% (Group 9). These groups typically feature domain-specific synonyms (Group 6) or label-based constraints (Group 9) that may not map cleanly onto each tool's internal dictionary or require morphological expansions not adequately covered by the default models.

Table 2. Entity Recognition Accuracy (%) by Group and Configuration

Grupo	DP	DP+F	S	DP+S	F+S	DP+F+S
0	28.50	46.97	62.53	65.96	80.74	**82.32**
1	49.72	64.25	68.16	76.54	83.24	**84.92**
2	36.97	54.51	66.27	69.68	**76.66**	**76.66**
3	34.18	44.30	75.32	79.11	**94.30**	**94.30**
4	25.96	39.37	52.93	57.31	63.44	**65.01**
5	50.52	71.28	62.47	70.44	79.04	**81.76**
6	7.10	13.66	36.07	37.70	42.08	**42.62**
7	75.22	87.61	87.61	92.92	94.69	**96.76**
8	46.25	68.75	75.00	83.75	96.25	**97.50**
9	0.00	2.26	46.79	46.79	**53.96**	**53.96**
10	80.25	88.89	79.01	92.59	93.83	**95.06**
11	17.72	24.68	48.73	50.63	54.43	**55.06**
12	53.38	70.27	66.22	76.35	81.76	**84.46**

Discussion. Entity recognition emerges as the pipeline's strongest step overall, but certain specialized or ambiguous queries remain a challenge. The collaboration among DeepPavlov (flexible entity recognition), Falcon (rule-based linking for short tokens), and SpaCy (robust parsing) lifts the average entity accuracy considerably. However, Groups like 6 (often multi-hop with ordering) and 11 (repeated references and textual filters) reveal persistent issues when the question's phrasing deviates from standard synonyms or tokens.

4.3 Property Extraction in Each Group

Table 3 presents the property extraction accuracy for each group. Like entity recognition, the metric indicates the proportion of queries in which *all* required properties were correctly detected and linked.

Key Observations. Whereas entity accuracy often climbs above 70–80%, property extraction remains below 60% in most groups. Groups 1 and 8 do exceed 80% property correctness under the best setup, suggesting that straightforward "double-check" relationships (Group 1) or multi-answer queries (Group 8) can be resolved if the lexical overlap is reasonably direct. Still, advanced queries in Groups 6, 9, and 11 again stand out as problematic. Partial coverage improvements (e.g., from 0.75% to 7.17% in Group 9) underscore that text-based property references remain an Achilles' heel when synonyms or domain-specific phrases are not captured by default extraction modules.

Table 3. Property Extraction Accuracy (%) by Group and Configuration

Grupo	DP	DP+F	S	DP+S	F+S	DP+F+S+Q
0	28.75	36.74	28.75	51.76	36.74	**67.09**
1	73.33	75.00	73.33	85.00	75.00	**86.67**
2	19.25	31.35	19.25	33.39	31.35	**52.47**
3	31.01	34.81	31.01	65.19	34.81	**74.68**
4	28.09	38.68	28.09	42.08	38.68	**54.92**
5	45.18	51.68	45.18	49.58	51.68	**55.97**
6	10.47	12.40	10.47	18.99	12.40	**24.42**
7	39.68	46.90	39.68	43.22	46.90	**51.18**
8	50.63	63.75	50.63	71.88	63.75	**84.38**
9	0.75	0.75	0.75	5.66	0.75	**7.17**
10	51.85	59.26	51.85	55.56	59.26	**66.67**
11	17.72	18.99	17.72	29.11	18.99	**32.91**
12	31.08	41.44	31.08	37.84	41.44	**50.90**

Reverse Lookups and Qualifiers. Leveraging reverse SPARQL lookups helps broaden the candidate property set, sometimes boosting coverage by 10–25% in multi-hop or filter-laden queries (Groups 2, 4, 5, 12). Qualifier constraints further refine the pipeline's ability to handle temporal or contextual edges (Groups 3, 7), preventing incomplete placeholders. Nonetheless, if a question's phrasing diverges from known synonyms—or the question includes nested multi-hop logic—misalignments remain common.

4.4 Final SPARQL Completeness

Definition of Completeness. The final metric measures how many queries matched the gold SPARQL *exactly* in terms of entities, properties, filters, and structure. Table 4 showcases the best setup's completeness values, which vary widely depending on how many pieces must align. Missing a single property or entity is enough to spoil the final match.

Table 4. Correct Entities, Properties, and SPARQL queries found (**DeepPavlov** + **Falcon** + **SpaCy** + qualifiers + reverse lookups)

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**

Group-Level Patterns. Groups 3 (counting) and 8 (multi-answer) reach above 70% completeness in the best-case scenario, reflecting that once the pipeline identifies the correct subject or object references, it can fill placeholders consistently. More challenging sets, like Groups 4, 5, 7, 9, 11, 12, rarely exceed 20% completeness, indicating that partial success in entity or property extraction does not yield a final SPARQL alignment. In Group 6, no queries are fully correct under any setup, reinforcing the complexity of queries with ordering, multiple relationships, or ambiguous references.

4.5 Exemplifying Challenges in Complex Queries

Despite the integration of multiple entity and property extraction tools, certain complex queries continue to pose significant challenges. This limitation becomes particularly evident when examining specific examples from challenging query groups, such as Group 6. Consider the following query from the LC-QUAD 2.1 dataset:

Question: *"Which Statkraft hydroelectric power station has the greatest annual energy output?"*

The Hereditary Tree-LSTM (HTL) successfully identified the appropriate dummy SPARQL template structure:

```
SELECT ?ent WHERE {
  ?ent DUMMY_P DUMMY_O .
  ?ent DUMMY_P ?obj .
  ?ent DUMMY_P DUMMY_O
} ORDER BY DESC(?obj) LIMIT DUMMY_F
```

The best possible SPARQL query that accurately addresses this question is:

```
SELECT ?ent WHERE {
  ?ent wdt:P31 wd:Q15911738 . # instance of
      hydroelectric power station
  ?ent wdt:P4141 ?obj . # annual energy
      output
  ?ent wdt:P127 wd:Q1681145 # owned by
      Statkraft
} ORDER BY DESC(?obj) LIMIT 1
```

However, the generated SPARQL query from the current pipeline resulted in inaccuracies:

```
SELECT ?ent WHERE {
  ?ent wdt:P31 wd:Q15911738 . # Correctly
      identified instance type
  ?ent wdt:P4131 ?obj . # Incorrect property
       (should be P4141 - annual energy
      output)
  ?ent wdt:P127 wd:Q1681145 # Correctly
      identified ownership by Statkraft
} ORDER BY DESC(?obj) LIMIT 1
```

In this specific case, the error lies in the incorrect identification of the property: the system selected property **P4131**, which refers to an unrelated attribute, whereas the correct property should be **P4141**. These inaccuracies arise mainly due to insufficient semantic alignment and ambiguities within the entity and property recognition modules, which struggle particularly with numeric or specific domain properties. Addressing such issues requires improved domain-specific embeddings, advanced morphological analysis, and refined ranking algorithms that can better capture subtle semantic distinctions inherent in complex queries.

Filtering Outliers. If we remove subsets of queries with extremely low coverage (e.g., Group 6 or Group 9), the final completeness among remaining groups rises significantly. This highlights how certain domain-specific or morphological details hamper the pipeline's property recognition enough

to break the entire SPARQL chain. Integrating more advanced morphological expansions and property disambiguation heuristics might improve these outlier groups in the future.

4.6 Discussion and Future Directions

Overall, these experiments confirm that combining multiple extraction tools (**DP+F+S+Q**) markedly improves entity and property coverage across most LC-QUAD 2.1 groups, compared to single-tool or pairwise settings. Nevertheless, advanced queries in Groups 6, 9, and 11, among others, expose how lexical variations, rarely used properties, and nested filters can still confound the pipeline. Specific directions for improvement include:

- **Domain-Specific Embeddings.** Incorporating custom embeddings or fine-tuning on domain text could capture synonyms or acronyms that default models miss, especially in Groups with specialized phrasing (e.g., 6 or 9).
- **Enhanced Ranking.** Dedicating more robust property-ranking heuristics or neural ranking methods might help handle borderline property matches, which hamper the final SPARQL alignment.
- **Morphological/Tokenization Improvements.** Groups that embed entities in multiword expressions (e.g., 11) might benefit from specialized tokenization logic and morphological checks.
- **Handling Nested or Rare Properties.** Introducing an expanded knowledge base of synonyms or advanced "reverse" lookups beyond standard properties could reduce the gap in multi-hop or heavily constrained questions.

Despite lingering gaps, the pipeline's success in many medium-complexity queries (like Groups 3 and 8) underscores the effectiveness of dummy templates, synergy among multiple extraction modules, reverse lookups, and qualifier constraints. By systematically identifying each group's structural logic, the pipeline reliably matches user questions to the correct SPARQL form, as long as the underlying entity and property references can be recognized.

5 Conclusion

This paper introduced a comprehensive and modular pipeline for Knowledge Base Question Answering (KBQA) that addresses significant shortcomings in existing entity- and property-extraction tools. Our framework employs multiple strategies, such as reverse SPARQL lookups and qualifier constraints, to broaden the set of potential predicates beyond what standard modules (DeepPavlov, Falcon, and SpaCy) typically capture. Although these off-the-shelf tools handle many straightforward cases, they often overlook essential properties in multi-hop or context-laden queries, leading to incomplete SPARQL alignments. By merging refined extraction processes with structured dummy templates and dynamic slot filling, we demonstrated a notable improvement in covering complex relationships, even when questions impose constraints or reference specialized knowledge. These enhancements highlight the importance of effectively retrieving not just entities but also less prominent predicates, a task crucial for assembling accurate and thorough SPARQL queries.

Despite these advances, there remain questions that involve rare or domain-specific properties, subtle morphological variations, or multiple layers of filtering and temporal qualifiers. Our results indicate that while the pipeline substantially raises recall in many scenarios, certain specialized or niche aspects still pose difficulties. Future research could focus on embedding-based expansions to better handle domain-specific synonyms, more refined morphological handling for short or compound tokens, and improved ranking models that weigh contextual information more robustly. The pipeline's modular design, anchored by interchangeable extraction and ranking components, should facilitate iterative improvements in these directions. Ultimately, this work underscores the need to look beyond entity identification and toward a richer, more adaptable strategy for retrieving properties, which is key for ensuring that KBQA systems can accurately address the intricacies and breadth of real-world queries.

Declarations

Authors' Contributions

Rômulo Chrispim de Mello: Conceptualization, Methodology, Software, Writing–original draft. **Jorão Gomes Jr**.: Writing–review & editing, Supervision. **Victor Ströele**: Writing–review & editing, Supervision. **Jairo Francisco de Souza**: Writing–review & editing, Supervision.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The original **LC-QUAD 2.0** dataset, containing over 30,000 natural language questions paired with SPARQL queries for DBpedia 2018 and Wikidata, is hosted at `https://github.com/AskNowQA/LC-QuAD2.0`. This version may contain duplicated entries and some misalignments. An updated and cleaned release, referred to as **LC-QUAD 2.1**, can be accessed on Zenodo at `https://zenodo.org/record/5508297`, which removes duplications, corrects mismatches, and normalizes question lengths. Last access to the research materials: 16 August 2025.

References

Cao, S., Shi, J., Pan, L., Nie, L., Xiang, Y., Hou, L., Li, J., He, B., and Zhang, H. (2022). KQA pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In Muresan, S., Nakov, P., and Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6101–6119,

Dublin, Ireland. Association for Computational Linguistics. DOI: https://doi.org/10.18653/v1/2022.acl-long.422.

Chen, S., Liu, Q., Yu, Z., Lin, C.-Y., Lou, J.-G., and Jiang, F. (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*, pages 325–336.

Daull, X., Bellot, P., Bruno, E., Martin, V., and Murisasco, E. (2023). Complex qa and language models hybrid architectures, survey. *arXiv preprint arXiv:2302.09051*.

DeepPavlov Team (2018□2024). Entity linking — deeppavlov 0.14.1 documentation. `https://docs.deeppavlov.ai/en/0.14.1/features/models/entity_linking.html`. Access on 16 August 2025.

Dileep, A. K., Mishra, A., Mehta, R., Uppal, S., Chakraborty, J., and Bansal, S. K. (2021). Template-based question answering analysis on the lc-quad2.0 dataset. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 443–448. DOI: https://doi.org/10.1109/ICSC50631.2021.00079.

Dubey, M., Banerjee, D., Abdelkawi, A., and Lehmann, J. (2019). Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *International Semantic Web Conference*, pages 69–78. Springer.

Dutt, R., Khosla, S., Bannihatti Kumar, V., and Gangadharaiah, R. (2023). GrailQA++: A challenging zero-shot benchmark for knowledge base question answering. In Park, J. C., Arase, Y., Hu, B., Lu, W., Wijaya, D., Purwarianti, A., and Krisnadhi, A. A., editors, *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 897–909, Nusa Dua, Bali. Association for Computational Linguistics. DOI: https://doi.org/10.18653/v1/2023.ijcnlp-main.58.

Ferragina, P. and Scaiella, U. (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*, CIKM '10, page 1625–1628, New York, NY, USA. Association for Computing Machinery. DOI: https://doi.org/10.1145/1871437.1871689.

Gomes, J., de Mello, R. C., Ströele, V., and de Souza, J. F. (2022). A hereditary attentive template-based approach for complex knowledge base question answering systems. *Expert Systems with Applications*, 205:117725. DOI: https://doi.org/10.1016/j.eswa.2022.117725.

Gomes Jr, J., de Mello, R. C., Ströele, V., and de Souza, J. F. (2022). A study of approaches to answering complex questions over knowledge bases. *Knowledge and Information Systems*, 64(11):2849–2881.

Gomes Jr., J., de Mello, R. C., Ströele, V., and de Souza, J. F. (2021). Lc-quad 2.1.

Hu, X., Wu, X., Shu, Y., and Qu, Y. (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*, pages 1687–1696, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Jang, H., Oh, Y., Jin, S., Jung, H., Kong, H., Lee, D., Jeon, D., and Kim, W. (2017). Kbqa: Constructing structured query graph from keyword query for semantic search. In *Proceedings of the International Conference on Electronic Commerce*, ICEC '17, New York, NY, USA. Association for Computing Machinery. DOI: https://doi.org/10.1145/3154943.3154955.

Jiménez, G., Leme, P. P., and Casanova, T. I. N. (2022). A framework to compute entity relatedness in large rdf knowledge bases. *Journal of Information and Data Management*, 13(2). DOI: https://doi.org/10.5753/jidm.2022.2435.

Jiménez, J., Leme, L., and Casanova, M. (2021a). Coepinkb: A framework to understand the connectivity of entity pairs in knowledge bases. In *Anais do XLVIII Seminário Integrado de Software e Hardware*, pages 97–105, Porto Alegre, RS, Brasil. SBC. DOI: https://doi.org/10.5753/semish.2021.15811.

Jiménez, J. G., Leme, L. P. P., Izquierdo, Y. T., Neves, A. B., and Casanova, M. (2021b). A distributed framework to investigate the entity relatedness problem in large rdf knowledge bases. In *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados*, pages 121–132, Porto Alegre, RS, Brasil. SBC. DOI: https://doi.org/10.5753/sbbd.2021.17871.

Lan, Y., He, G., Jiang, J., Jiang, J., Zhao, W. X., and Wen, J. (2021). A survey on complex knowledge base question answering: Methods, challenges and solutions. *CoRR*, abs/2105.11644.

Mello, R., Jr., J. G., Souza, J., and Ströele, V. (2024). Constructing a kbqa framework: Design and implementation. In *Proceedings of the 30th Brazilian Symposium on Multimedia and the Web*, pages 89–97, Porto Alegre, RS, Brasil. SBC. DOI: https://doi.org/10.5753/webmedia.2024.243150.

Mihindukulasooriya, N., Rossiello, G., Kapanipathi, P., Abdelaziz, I., Ravishankar, S., Yu, M., Gliozzo, A., Roukos, S., and Gray, A. G. (2020). Leveraging semantic parsing for relation linking over knowledge bases. *CoRR*, abs/2009.07726.

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

Perevalov, A., Diefenbach, D., Usbeck, R., and Both, A. (2022). Qald-9-plus: A multilingual dataset for question answering over dbpedia and wikidata translated by native speakers. *CoRR*, abs/2202.00120.

Qin, K., Wang, Y., Li, C., Gunaratna, K., Jin, H., Pavlu, V., and Aslam, J. A. (2020). A complex KBQA system using multiple reasoning paths. *CoRR*, abs/2005.10970.

Rolim, T., Avila, C. V., Junior, N. A., Costa, F., Mariano, R., Calixto, T., and Vidal, V. M. (2021). Kg-e: Um grafo de conhecimento semântico baseado na integração de dados de empresas e sancionados. In *Anais do IX Workshop de Computação Aplicada em Governo Eletrônico*, pages 155–166, Porto Alegre, RS, Brasil. SBC. DOI: https://doi.org/10.5753/wcge.2021.15985.

Rossiello, G., Mihindukulasooriya, N., Abdelaziz, I., Bornea, M., Gliozzo, A., Naseem, T., and Kapanipathi, P. (2021). Generative relation linking for question answering over knowledge bases. In Hotho, A., Blomqvist, E., Dietze, S., Fokoue, A., Ding, Y., Barnaghi, P., Haller, A., Dragoni, M., and Alani, H., editors, *The Semantic Web – ISWC 2021*, pages 321–337, Cham. Springer International Publishing.

Sakor, A., Singh, K., Patel, A., and Vidal, M. (2019). FALCON 2.0: An entity and relation linking tool over wikidata. *CoRR*, abs/1912.11270.

Sorokin, D. and Gurevych, I. (2018). Modeling semantics with gated graph neural networks for knowledge base question answering. In Bender, E. M., Derczynski, L., and Isabelle, P., editors, *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3306–3317, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Tan, Y., Min, D., Li, Y., Li, W., Hu, N., Chen, Y., and Qi, G. (2023). Evaluation of chatgpt as a question answering system for answering complex questions. *arXiv preprint arXiv:2303.07992*.

Trivedi, P., Maheshwari, G., Dubey, M., and Lehmann, J. (2017). Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*, pages 210–218, Cham. Springer, Springer International Publishing.

Usbeck, R., Ngomo, A.-C. N., Haarmann, B., Krithara, A., Röder, M., and Napolitano, G. (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*, pages 59–69. Springer.

Xie, Z., Zeng, Z., Zhou, G., and He, T. (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*, pages 300–311. Springer.

Ye, X., Yavuz, S., Hashimoto, K., Zhou, Y., and Xiong, C. (2021). Rng-kbqa: Generation augmented iterative ranking for knowledge base question answering. *arXiv preprint arXiv:2109.08678*.