Karlsruhe Institute of Technology

Software Design and Quality

HubLink: Leveraging Language Models for Enhanced Scholarly Information Retrieval on Research Knowledge Graphs

Master's Thesis of

Marco Schneider

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

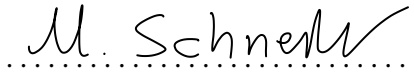
First examiner:	Prof. Dr. Ralf H. Reussner
Second examiner:	Prof. Dr.-Ing. Anne Koziolek
First advisor:	Dipl.-Inform. Angelika Kaplan
Second advisor:	Dr.-Ing. Jan Keim

30. November 2024 – 30. May 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 30.05.2025



(Marco Schneider)

Abstract

The exponential growth of scholarly literature presents significant challenges for researchers as the search for relevant scholarly contributions is a time-consuming and cognitively taxing process. Although the integration of Research Knowledge Graphs (RKGs) with Question Answering over Knowledge Graphs (KGQA) systems is a promising alternative to the current document-centric communication of scholarly knowledge, current approaches often face limitations. Most current approaches for retrieving data from RKGs are based on Semantic Parsing (SP), which faces practical difficulties due to a dependence on the underlying graph schema and the need for training data. This leaves the potential of schema-agnostic, training-free KGQA approaches largely underexplored in the scholarly domain. Furthermore, the systematic evaluation of KGQA retrieval capabilities is hindered by the absence of a comprehensive taxonomy tailored to classify questions specific to scholarly literature search tasks.

This thesis addresses these gaps by presenting HubLink, a novel schema-agnostic and training-free approach designed for scholarly KGQA. HubLink utilizes pre-trained Large Language Models (LLMs) and operates by conceptually decomposing the RKG into *hubs*, which aggregate knowledge from individual publications. This structure facilitates source-aware inference, thereby enhancing transparency and provenance tracking, which are crucial for scholarly tasks. This is achieved without relying on fixed schemas or training datasets. To support robust evaluation, this work also introduces a taxonomy for classifying scholarly KGQA questions, enabling detailed evaluation of KGQA system characteristics and capabilities. The development of this taxonomy followed an operationalized and replicable construction process, which is also proposed in this thesis. Based on the generated taxonomy, new KGQA benchmarking datasets for the Open Research Knowledge Graph (ORKG) have been constructed, featuring diverse query types, specific scholarly literature search use cases, and multiple graph schema variations.

Evaluation against five state-of-the-art baseline methods demonstrates that the HubLink approach achieves superior performance in retrieving relevant context, particularly for complex questions related to literature search use cases, thereby highlighting limitations in current methods and the advancements offered by this work. By providing both a novel KGQA approach and benchmarking tools, this thesis provides a significant contribution towards advancing scholarly KGQA, paving the way for more efficient, reliable, and knowledge-centric scientific communication.

Zusammenfassung

Das exponentielle Wachstum wissenschaftlicher Literatur stellt Forschende vor erhebliche Herausforderungen, da die Suche nach relevanten wissenschaftlichen Beiträgen ein zeitaufwändiger und kognitiv anspruchsvoller Prozess ist. Obwohl die Integration von Research Knowledge Graphs (RKGs) mit Question Answering over Knowledge Graphs (KGQA) Systemen eine vielversprechende Alternative zur derzeitigen dokumentenzentrierten Kommunikation wissenschaftlicher Erkenntnisse darstellt, stoßen aktuelle Ansätze oft an Grenzen. Die meisten aktuellen Ansätze zur Datenextraktion aus RKGs basieren auf Semantic Parsing (SP), was aufgrund der Abhängigkeit vom zugrundeliegenden Graphschema und dem Bedarf an Trainingsdaten auf praktische Schwierigkeiten stößt. Dies lässt das Potenzial schema-agnostischer und trainingsfreier KGQA-Ansätze für wissenschaftliche Literatursuche weitgehend ungenutzt.

Diese Arbeit schließt diese Lücke, indem sie HubLink vorstellt, einen schema-agnostischen und trainingsfreien Ansatz, der für KGQA entwickelt wurde. HubLink verwendet vortrainierte Large Language Models (LLMs) und unterteilt den RKG konzeptionell in *Hubs*, die Wissen aus einzelnen Publikationen aggregieren. Diese Struktur ermöglicht eine quellenbewusste Inferenz, wodurch die für wissenschaftliche Aufgaben entscheidende Transparenz und Nachverfolgbarkeit der Datenherkunft verbessert wird, ohne auf feste Schemata oder Trainingsdatensätze angewiesen zu sein. Zur Unterstützung einer robusten Evaluierung führt diese Arbeit außerdem eine Taxonomie zur Klassifizierung wissenschaftlicher KGQA-Fragen ein, die eine detaillierte Bewertung der Eigenschaften und Fähigkeiten von KGQA-Systemen ermöglicht. Die Entwicklung dieser Taxonomie folgte einem operationalisierten und replizierbaren Erstellungsprozess, der ebenfalls in dieser Arbeit vorgeschlagen wird. Basierend auf der erstellten Taxonomie wurden neue KGQA-Benchmarking-Datensätze für den Open Research Knowledge Graph (ORKG) erstellt, die vielfältige Anfragetypen, spezifische Anwendungsfälle der wissenschaftlichen Literatursuche und mehrere Graphschema-Variationen umfassen.

Die Evaluierung anhand von fünf State-of-the-Art-Baseline-Methoden zeigt, dass der HubLink-Ansatz eine überlegene Leistung bei der Extraktion von relevanten Kontexten erzielt, insbesondere bei komplexen Fragen im Zusammenhang mit Anwendungsfällen der Literatursuche. Dies unterstreicht die Grenzen aktueller Methoden und die Fortschritte, die diese Arbeit bietet. Durch die Bereitstellung sowohl eines neuen KGQA-Ansatzes als auch von Benchmarking-Werkzeugen leistet diese Masterarbeit einen wesentlichen Beitrag zur Weiterentwicklung im Bereich von KGQA für wissenschaftliche Daten und ebnet den Weg für eine effizientere, zuverlässigere und wissenszentrierte wissenschaftliche Kommunikation.

Contents

List of Figures

List of Tables

Acronyms

ANN Approximate Nearest Neighbor.

API Application Programming Interface.

BLEU Bilingual Evaluation Understudy.

CE Carbon Emissions.

CLI Command Line Interface.

DiFaR Direct Fact Retrieval.

DOI Digital Object Identifier.

DSR Design Science Research.

ECSA European Conference on Software Architecture.

ENN Exact Nearest Neighbor.

EO Evaluable Output.

FN False Negative.

FP False Positive.

GNN Graph Neural Network.

GQM Goal Question Metric.

GraphRAG Graph Retrieval Augmented Generation.

GT Ground Truth.

ICSA International Conference on Software Architecture.

IR Information Retrieval.

IRI Internationalized Resource Identifier.

IS Information System.

KARAGEN Knowledge Augmented Retrieval and Generation ENgine.

KB Knowledge Base.

KBQA Question Answering over Knowledge Bases.

KG Knowledge Graph.

KGQA Question Answering over Knowledge Graphs.

LLM Large Language Model.

OFAT One-Factor-at-a-Time.

ORKG Open Research Knowledge Graph.

QA Question Answering.

RAG Retrieval-Augmented Generation.

RDF Resource Description Framework.

RGAR Retrieval, Generation, and Additional Requirement.

RKG Research Knowledge Graph.

ROUGE Recall-Oriented Understudy for Gisting Evaluation.

SE Software Engineering.

SP Semantic Parsing.

SQA Scholarly Question Answering.

SWA Software Architecture.

TN True Negative.

ToG Think-on-Graph.

TP True Positive.

1. Introduction

Today, almost all scholarly findings are communicated through digital scholarly articles, a practice often referred to as the document-centric approach [1, 2]. Although the transition from print to digital has significantly improved accessibility [3, 4], it still does not fully take advantage of modern digital technologies. In current research practice, each publication is treated as an isolated unit of knowledge, making it challenging to interlink related findings, methodologies, and underlying data. This isolation becomes especially problematic given the exponential growth of scientific literature [5]. Researchers must navigate an extensive body of text to identify relevant insights. This process is challenging due to the sheer volume of content and also the addition of inherent issues that plague text searches. For example, the lexical gap problem, in which misspellings, synonyms, abbreviations, ambiguous words, and the ignored word order hinder the effectiveness of searches [6]. Consequently, literature search becomes a time-consuming and cognitively taxing process, requiring careful crafting of search queries, manual sifting through countless documents, and painstaking synthesis of scattered evidence.

Recently, Large Language Models (LLMs) have demonstrated powerful capabilities in understanding natural language, expressiveness, general knowledge, and generalization [7]. In the context of Question Answering (QA), integrating LLMs allows users to pose queries in natural language, with the system generating relevant answers. This is particularly interesting for scholarly literature search as it allows researchers to find answers to scientific resources by asking questions in natural language, effectively reducing manual effort. However, using the pre-trained internal knowledge of the LLMs is not sufficient because their answers may lack depth, be subject to hallucinations, and do not provide transparency regarding the underlying reasoning process [7].

To mitigate these issues in the QA setting, Retrieval-Augmented Generation (RAG) has emerged, in which an external Knowledge Base (KB) is used to provide relevant context to the LLM to enhance the answer generation [8]. In this framework, an indexing phase takes place where the collection of documents is split into text chunks, which are then converted into dense vector representations and stored in a vector store. At query time, the vector representation of the question is used to conduct a distance-based search in the vector space to find the most relevant text chunks. Although this approach can be applied to the scholarly literature search setting [9], naive RAG encounters notable drawbacks. Retrieval can suffer from poor precision and recall, missing crucial context, or retrieving irrelevant information [10]. Furthermore, LLMs often struggle with noisy or misleading documents, fail to abstain from answering when relevant information is absent, perform poorly at integrating information across multiple sources, and are prone to accepting factual errors in the retrieved context [11].

An alternative to document-centric knowledge representation is to structure and interconnect representations of scholarly knowledge [1]. This can be achieved using Knowledge Graphs (KGs) where knowledge is curated from structured and unstructured sources [12]. Specifically for the academic domain, these graphs are referred to as Research Knowledge Graphs (RKGs) and offer a promising solution to structure and interlink scholarly information. With the aim of transforming scientific communication from document-centric to knowledge-centric [2], RKGs provide a structured and machine-readable representation of scholarly knowledge by taking into account the relationships between texts and the incorporation of structural information as additional knowledge [13].

The unification of LLMs and KGs is a relatively new research field that aims to utilize the advantages of both in various applications [14]. One such application is Question Answering over Knowledge Graphs (KGQA), where the integration of knowledge from a KG to inform the LLM during inference is investigated [15, 16, 17, 14, 18, 13, 19, 20, 21, 22, 23]. Although many KGQA approaches have emerged, most of the approaches are focused on open-domain QA and have not yet been tested in the scholarly field. Furthermore, approaches explicitly aimed at the scientific field focus primarily on Semantic Parsing (SP) methods that automatically create structured queries from natural language queries using training examples [24, 25, 26, 27, 28].

Despite being effective in controlled settings, SP methods struggle to adapt to dynamically evolving KG schemas, leading to poor performance when encountering previously unseen entities or relations. In addition, semantic parsing approaches face significant scalability challenges, including both schema-level and fact-level complexities [29]. These problems are exacerbated by the extensive schema richness of modern KGs, which often encompasses thousands of schema items. Moreover, semantic parsing methods typically require manually curated training datasets, which are costly to produce and limit their applicability in real-world research environments, making the methods inconvenient to use.

Consequently, we identify a research gap in applying alternative LLM-based strategies to the scholarly KGQA task. Most alternative strategies are developed and evaluated on general-purpose graphs like Freebase or Wikidata [13]. However, these approaches cannot be easily adapted to the scholarly domain [30]. Moreover, current retrieval approaches often ignore the source diversity of retrieved information, potentially returning multiple pieces of information from the same publication, reducing the breadth of evidence needed for robust scholarly inquiries. Therefore, the scholarly domain remains underexplored within KGQA research.

Motivated by these challenges, we argue that research on scholarly KGQA should explore directions beyond semantic parsing to develop methods that do not require training or fine-tuning and can adapt to dynamically changing KG schemas. Our thesis contributes a novel KGQA retrieval approach that complies with the above-mentioned criteria. We rigorously evaluated the approach on a scholarly literature search task in a Software Architecture (SWA) setting against five state-of-the-art baseline KGQA approaches on the Open Research Knowledge Graph (ORKG) to demonstrate its superiority in retrieval performance. We further contribute a new taxonomy for the classification of questions querying KGQA

systems for literature search. With these contributions, our aim is to advance research in the direction of KGQA in the scholarly literature search domain.

1.1. Problem Statements

KGQA approaches show promising results in the QA setting. However, most approaches are applied to the general-purpose domain and are not tailored to scholarly content. Those approaches that target scholarly KGQA often rely on SP techniques that struggle with dynamically evolving schemas and often require training data. As a result, the potential of applying KGQA to support effective scholarly search remains largely untapped, leading to our first problem statement:

P1 There is an underexplored potential in applying KGQA in the scholarly domain to help researchers find relevant literature faster and more reliably. Initial approaches show promising results, but they struggle with evolving schemas and often require training data. This problem hinders the practical application of QA systems on the scholarly literature search task in real-world scenarios.

The application of a KGQA approach to an RKG has the potential to help researchers in finding related literature faster and more reliably. This is made possible through the natural language capabilities of LLMs that allow researchers to ask for information of interest using natural language rather than having to manually search for it.

To assess whether such a KGQA system is capable of answering desirable questions for literature search, a taxonomy can help. Such a taxonomy classifies questions according to their complexity and the retrieval capabilities required to arrive at the answer. Although there are existing taxonomies, such as in DBLP-QuAD [24] and SciQA [31] for KGQA or the works provided by Li and Roth [32] and Singhal et al. [33] for general QA, they often propose divergent classification schemes. Without a synthesized and domain-relevant taxonomy, it is challenging to assess whether a given system can handle the full range of question types and retrieval complexities found in scholarly search tasks. Such a taxonomy can be helpful to guide the development of robust KGQA datasets used for benchmarking and tailored to the needs of researchers, leading to our second problem statement:

P2 There is a lack of a taxonomy that allows the classification of the characteristics of questions posed to KGQA retrieval systems in the scholarly literature search task. This hinders the creation of diverse question datasets to test the retrieval capabilities of KGQA approaches on different RKG.

Using such a taxonomy and applying it to the scholarly literature search task can help in the process of creating QA datasets to reliably determine whether a retrieval approach is able to meet its challenges.

1.2. Objective and Research Questions

After discussing the potential benefits and importance of utilizing KGQA to enhance the search of scholarly literature and addressing the issues with existing approaches (**P1**) together with the evaluation of KGQA retrieval systems (**P2**), we establish the research objective of this thesis as follows:

> **Research Goal:** Design a training-free and schema-agnostic retrieval approach leveraging RKGs and pre-trained LLMs that enhances the quality, transparency, and reliability of context retrieval in KGQA systems specifically tailored to scholarly literature search tasks. Additionally, construct a taxonomy for classifying questions posed to KGQA retrieval systems to support the assessment of capabilities and performance of such systems in the scholarly literature task.

A new retrieval approach that does not rely on a KG schema and is training-free would benefit the exploration of the potential that such systems can have in literature search tasks. This addresses the first problem, **P1**. Furthermore, a taxonomy for characterizing questions in KGQA retrieval can help to guide the construction of KGQA datasets used for benchmarking retrieval systems. This addresses the second problem, **P2**.

To achieve this goal, the thesis is guided by research questions. The first question concerns the design and technical foundation of the new KGQA retrieval approach:

RQ1 How can a schema-agnostic retrieval algorithm leveraging an RKG and a pre-trained LLM be developed for the KGQA setting to effectively integrate diverse scholarly sources, adapt to evolving schemas and account for the provenance of information during retrieval without relying on training data?

This research question addresses the first problem (**P1**) as it directly targets the limitations of current scholarly KGQA approaches. By developing a schema-agnostic and training-free approach, it explores the potential of retrieval without relying on the schema of the RKG and thus is scalable and dynamically adaptable. Furthermore, without the requirement of a training dataset, the approach becomes more conveniently applicable and thus more relevant to apply in a real-world use case. Moreover, accounting for the provenance of information during inference is a critical aspect of scholarly literature search, as it supports the generation of complete, transparent, and balanced results. Provenance awareness helps prevent over-reliance on a single source and encourages the integration of diverse perspectives, ensuring that results capture complementary aspects that may not be covered by only one source alone.

The second question is related to understanding the nature of questions targeting scholarly KGQA systems. Since the search for scholarly literature covers a wide range of information needs, it is important to characterize these needs and ensure that the evaluation datasets reflect their diversity, which can be achieved using a taxonomy. This motivates the following question:

RQ2 How can existing general QA and KGQA taxonomies be synthesized and extended to form a comprehensive taxonomy tailored to define the characteristics of questions posed to KGQA retrieval systems for the literature search task?

The second research question addresses the second problem (**P2**). A taxonomy of this kind offers insight into the extent of capabilities that a KGQA approach has in answering questions posed with regard to the scholarly literature search task. In our thesis, we apply this taxonomy to create KGQA datasets that allow the evaluation of our proposed KGQA approach and understand its capabilities.

1.3. Research Contributions

This thesis provides the following contributions to investigate the potential of using KGQA to support scholarly literature search:

C1 A novel KGQA retrieval approach named *HubLink*, which is a schema-agnostic and training-free method designed to conduct source-aware inference on KGs.

C1.1 The Scholarly Question Answering (SQA)-framework, which enables modular testing of RAG pipelines with various KGs and retrieval approaches and the semi-automatic generation of KGQA datasets.

C1.2 Implementations of five training-free and schema-agnostic KGQA retrieval approaches from the existing literature adapted for use on the ORKG.

C2 A question taxonomy for classifying questions targeting scholarly KGQA, facilitating the development of diverse KGQA datasets to evaluate the performance and capabilities of retrieval systems.

C2.1 A systematic question taxonomy construction process to synthesize existing knowledge from the literature, emphasizing replicability, transparency, and validation.

C3 A new KGQA dataset for the ORKG featuring four variants for different graph schemas, designed to benchmark the performance and robustness of KGQA systems across a wide range of question types in the scholarly literature search task.

The primary contribution of this thesis is HubLink (**C1**), a novel KGQA approach that conceptually decomposes the graph into structures termed *hubs*. Each hub is individually evaluated to determine its relevance in answering the provided question, with a subsequent generation of partial answers for each relevant hub. These partial answers are then synthesized into a final answer. This modular approach enables the retrieval process to be source-aware, schema-agnostic, and training-free. To support this contribution, a framework (**C1.1**) that implements a modular and configurable RAG pipeline to benchmark KGQA approaches on different KGs is contributed. Subsequently, implementations of five established state-of-the-art KGQA approaches from the literature are provided, which were

previously tested only in open-domain settings. Their implementations were adapted (**C1.2**) for compatibility with the ORKG and tested with regard to their performance against our novel HubLink approach in an extensive evaluation.

Furthermore, a question taxonomy (**C2**) is contributed, which enables the classification of questions specifically targeting KGQA for the scholarly literature search task. This facilitates the creation of diverse and complex KGQA datasets incorporating questions designed to test various capabilities of KGQA approaches. To support this contribution, a taxonomy construction process (**C2.1**) is provided that systematically synthesizes knowledge from the literature and operationalizes the generation of taxonomies, complete with tool support for easy application.

Finally, new KGQA datasets (**C3**) for the ORKG are contributed, which are based on an SWA schema and the question taxonomy (**C2**). Consequently, the datasets include a variety of questions that target different retrieval capabilities to thoroughly test KGQA approaches. In addition, the datasets are based on six distinct use cases for the scholarly literature search task to ensure relevance to real-world scenarios and relate to four different graph variants corresponding to different graph schemas, to evaluate the schema-agnostic property and robustness of KGQA approaches.

1.4. Thesis Outline

After the introduction, the remainder of the master thesis is structured as follows:

Chapter 2 establishes the fundamental concepts upon which the research is built. This chapter introduces KGs, with a specific focus on RKGs. It further explains the ORKG, which is the graph used to conduct our experiments on. The chapter then continues with an explanation of KGQA, the primary research field toward which this work is contributing. Furthermore, it includes an explanation of LLMs, covers Graph Retrieval Augmented Generation (GraphRAG), Approximate Nearest Neighbor (ANN) search, and a range of evaluation metrics relevant to retrieval and generation components. Finally, the chapter explains the concept of taxonomies, including their formal definition, development approaches, and evaluation methods.

Chapter 3 provides a comprehensive review of existing literature relevant to this thesis. It first examines KGQA approaches that utilize LLMs in the scholarly and open domains. The discussion then continues with taxonomy development and question classification, covering systematic approaches to taxonomy construction and evaluation, alongside question classification for scholarly QA on KGs. Finally, it examines the benchmarking of KGQA systems, including datasets targeting open-domain and scholarly domain KGs, and methodologies for QA dataset construction.

Chapter 4 presents our primary contribution of this thesis: *HubLink*. It provides an overview of the HubLink approach, detailing the indexing, retrieval, and generation phases. It then proceeds to explain the formal definitions of key concepts of the approach, before explaining

the approach in detail using pseudocode. In addition, the chapter outlines the challenges and design rationale for decisions made during the development of the approach. Finally, the chapter concludes with a discussion on the generalizability, scalability, index updating mechanisms, and limitations.

Chapter 5 outlines the systematic process proposed in this thesis for the creation of taxonomies by synthesizing knowledge from the literature. The chapter begins with an evaluation plan that utilizes an Goal Question Metric (GQM) model to facilitate the evaluation of a taxonomy. Subsequently, the iterative development process is outlined, which encompasses planning, literature survey, extraction of relevant concepts, clustering of these concepts, relevance assessment, the actual taxonomy construction and refinement stages, validation of the taxonomy, and finally, its application. The chapter also describes construction artifacts and discusses the limitations inherent in the proposed methodology.

Chapter 6 details the application of the proposed systematic taxonomy construction approach, by creating a KGQA retrieval taxonomy, aimed at the classification of questions for scholarly literature search. The chapter starts with the planning and a systematic literature survey, before describing the extraction of classes, including their distribution by category, domain, publication year, and citation metrics. Then, the clustering of extracted classes, including deduplication and categorization processes, is explained, followed by a relevance assessment of these clusters. Following this, the iterative construction and refinement of the taxonomy through three increments are presented. The chapter then systematically describes the categories and classes of the final taxonomy, before its application on SWA research questions is demonstrated.

Chapter 7 introduces the SQA framework, central for the implementation of the KGQA approaches, the creation of the KGQA datasets, and the execution of the experiments. Furthermore, the chapter details the implementation of HubLink and the baselines KGQA approaches, also describing the selection process by which the approaches have been chosen. The chapter further specifies the methods for accessing and populating the ORKG.

Chapter 8 sets the stage for the experimental evaluation of the HubLink approach against the baseline approaches. The chapter outlines the overall evaluation concept and describes the software and hardware environment used for the experiments. Furthermore, the creation of the KGQA datasets, detailing the use cases for scholarly literature search, an overview of content granularity, and the dataset creation process is described. Following this, the evaluation framework and metrics are specified, including the evaluation targets and a detailed evaluation plan using the GQM model.

Chapter 9 details the methodology and results of the parameter selection process, which has been applied to select the configurations for the HubLink approach and the selected baseline methods. The chapter begins with the planning of the process, before presenting the results for the approaches HubLink, DiFaR, FiDeLiS, and Mindmap, each including their base configuration, parameter ranges explored, and the final selected parameters.

Chapter 10 presents and discusses the results of the comprehensive evaluation of the HubLink approach against the baseline methods. The evaluation is divided into two main parts: evaluating retrieval quality and evaluating answer alignment. The retrieval quality

assessment examines the improvement of retrieval accuracy and relevance, the impact of operation complexity, applicability to different scholarly literature search use cases, the impact of type information in the question, robustness to structural and lexical variability in the graph schema, analysis of runtime and LLM token consumption, and the environmental sustainability impact. The answer alignment evaluation focuses on the semantical and factual consistency of generated answers, the generation of relevant answers, adherence to instructions provided in the question, and the consistency of generated answers with the retrieved context. The chapter concludes with a detailed discussion of the evaluation results.

Chapter 11 summarizes the key findings of the thesis and reflects on the research objectives and questions. It revisits the research questions posed at the beginning of the thesis, providing answers based on the research conducted and contributions made and concludes by discussing avenues for future work.

2. Foundations

In this section, we explain the foundational concepts of this master thesis. We begin with the explanation of KGQA and KGs. Then, we provide an overview of LLMs and describe how GraphRAG can unify the abilities of LLMs with the structured knowledge from KGs for answering natural language questions. Following this, we provide an explanation of ANN, a fundamental concept of our HubLink approach. Next, we describe the metrics that are commonly used in RAG research, which we will also be using in our evaluation. Lastly, we define the term taxonomy within the scope of the thesis and explain its construction and evaluation concepts.

2.1. Knowledge Graphs

Knowledge Graphs (KGs) represent information as structured networks. In these networks, nodes typically denote entities such as individuals, locations, publications, or abstract concepts, while the edges signify the relationships that connect these entities [15, 14]. Although there are numerous definitions for KGs, many prominent implementations are modeled using the Resource Description Framework (RDF) [34]. RDF provides a graph-based data model standardized by the World Wide Web Consortium (W3C) [35]. Consequently, for the scope of this thesis, the term KG will specifically refer to an RDF graph.

The fundamental unit of an RDF graph is the *triple*, a tuple comprising three components: a subject, a predicate, and an object [14, 35, 36]. Following the W3C standard, an RDF graph G is formally defined as a set of such triples:

$$G = \{\, t = (s, p, o) \mid s \in \mathcal{I} \cup \mathcal{B},\ p \in \mathcal{I},\ o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \,\},$$

where:

- $\mathcal{I}$ is the set of Internationalized Resource Identifiers (IRIs), which uniquely identify resources,
- $\mathcal{B}$ is the set of blank nodes representing resources without a global identifier,
- $\mathcal{L}$ is the set of literals, which are atomic pieces of information (e.g., strings, numbers).

Furthermore, a triple $t = (s, p, o)$ consists of:

- *Subject s*: The resource being described, which is either an IRI or a blank node.

- *Predicate p*: A property (or relation) represented as an IRI that links the subject to the object.
- *Object o*: The target or value of the relationship, which may be an IRI, a blank node, or a literal.

The key characteristics of a KG have been expressed by [12]:

- The structure of a KG depends on the use of an ontology, which defines the concepts, properties, and associations across a single or multiple domains.
- A KG is formulated in a triple format and is often stored as an RDF graph.
- To create the KG, knowledge is curated from structured and unstructured sources.
- The KG is stored in a graph database management system and queried using adaptive querying languages such as CYPHER, SPARQL, SQL, and API calls to retrieve data from the graph.

2.1.1. Research Knowledge Graphs

KGs are classified based on the information they store. Several categories are recognized in the literature [14]: *Encyclopedic KGs* aim to represent real-world knowledge, often integrating information from diverse sources like encyclopedias, expert input, and databases. *Commonsense KGs* focus on modeling tacit, everyday knowledge and the relationships between common concepts. *Domain-specific KGs* contain specialized information pertinent to particular fields. Furthermore, *Multi-modal KGs* extend the traditional graph structure to incorporate information from various modalities, including images, audio, and video.

Another type of KGs are Research Knowledge Graphs (RKGs), which are graphs focused on scholarly communication. Although an RKG can also store typical data such as people, documents, datasets, and institutions, it also includes relationships between problems, methods, and results. A distinguishing feature is the representation of statements extracted from scientific articles as semantic resources within the graph. These resources are explicitly linked to their source articles, enabling direct traceability of information provenance. [2]

Karras et al. [37] propose a categorization of RKGs into *generic* and *specific* types. Generic RKGs primarily utilize bibliographic metadata to structure entities and relationships. Prominent examples include the *Microsoft Academic Knowledge Graph* [38, 39], *OpenAlex* [40], *Springer Nature SciGraph* [41], *Semantic Scholar Literature Graph* [42], *OpenAIRE Research Graph* [43], *Research Graph* [44], and the *Scholarly Link Exchange (Scholix)* framework [45]. In contrast, specific RKGs focus on storing detailed scientific data in addition to bibliographic metadata. These graphs are employed to describe and link research artifacts and entities, often tailored to particular scientific topics or domains. Notable examples are *CovidGraph* [46], *SoftwareKG* [47], the *Computer Science Knowledge Graph (CS-KG)* [48], the *Cooperation Databank (CoDa)* [49], and *OpenBiodiv* [50].

Subject	Predicate	Object
{ Resource, Property, Class }	Property	{ Resource, Property, Class, Literal }

Table 2.1.: The permitted types in ORKG statements for the Subject, Predicate, and Object in a triple, based on [51, p. 21].

2.1.2. The Open Research Knowledge Graph (ORKG)

Beyond the generic and specific classifications, the Open Research Knowledge Graph (ORKG) presents a distinct approach by aiming to organize topic-specific scientific data semantically across diverse research domains [37].

The ORKG is an RKG that stores scientific articles along with their contributions as semantic units. It provides the infrastructure for acquiring, curating, publishing, and processing semantic scientific knowledge with the goal of making scientific findings more accessible to both humans and machines [1]. The graph is publicly accessible through a website[1] and is maintained by the ORKG community. New entries can be added by any user, with entry quality being monitored by administrators. At the time of writing this thesis, the graph contains over 33,800 articles across more than 700 different research fields[2].

Data Model Knowledge in the ORKG is stored closely following the RDF format, meaning that at the core, the graph is stored in triples, which are also referred to as *statements*. As detailed in Table 2.1, each component of a statement accommodates specific types of information. Subjects and objects can represent resources, properties, or classes, whereas predicates are restricted to properties. Literals are used for atomic data, such as text, numbers, or dates, and are used exclusively as objects within statements. Furthermore, the ORKG system automatically assigns a unique identifier (ID) to each resource, property, class, and literal instance. [51, pp. 21–22]

Content Types Information stored in the ORKG is organized using distinct content types. A primary content type is the *paper*. When a paper is added, the system automatically populates associated metadata, including its DOI, title, research field, authors, publication date, venue, and URL. Another crucial content type is the *contribution*, which allows users to add structured data describing the research findings presented in the paper, extending beyond simple metadata. These contributions can subsequently be utilized within *comparisons*, another ORKG content type. Comparisons facilitate the contrasting of research findings across multiple papers and can be semi-automatically generated by the system to identify shared properties among different contributions. [51, p. 22]

Contribution Templates To promote standardization in the way contributions are structured, the ORKG offers *templates*. These templates implement a subset of the Shapes Constraint Language (SHACL) [52] and define the expected structure for contributions

[1] `https://orkg.org/` [last accessed on 21.12.2024]
[2] `https://orkg.org/stats` [last accessed on 21.12.2024]

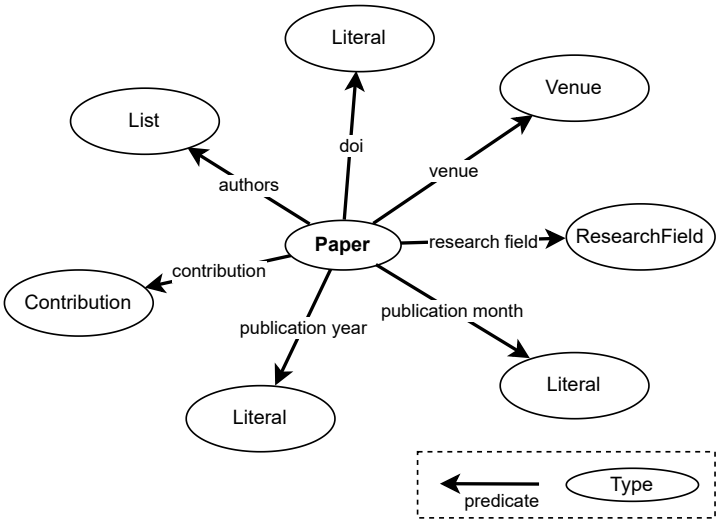


Figure 2.1.: Schematic representation of how publication data is structured in the ORKG, illustrating the root entity *Paper* and its linkage to metadata and *Contribution* entities.

related to specific types of research descriptions. Intended for creation by domain experts, templates specify constraints and requirements, ensuring consistency and quality in the data entered for contributions within particular research areas. [51, pp. 58–60]

Research Fields The concept of *research fields* serves as a primary organizational mechanism within the ORKG. Research fields can be associated with various content types (papers, contributions, comparisons, templates, etc.) to enable the grouping and categorization of related content within the graph. [51, pp. 29–30]

Lists *Lists* provide another organizational tool in the ORKG. They allow users to group related papers thematically or for specific purposes without necessarily requiring the addition of detailed structured data for each paper within the list. [51, pp. 28–29]

Storing Publications As previously mentioned, scientific publications are represented in the ORKG using the *paper* content type. Interaction with these paper entities forms the principal mode of engagement with the ORKG relevant to this thesis. Figure 2.1 depicts the organizational structure for storing publication data. The diagram illustrates that the paper entity serves as the central node for publication information. All associated data, including metadata and one or more contributions detailing the research content, are linked to this paper entity using statements.

2.1.3. The KARAGEN Approach for Question Answering on the ORKG

Knowledge Augmented Retrieval and Generation ENgine (KARAGEN) is an approach for implementing a QA system on the ORKG graph. The goal is to take advantage of the

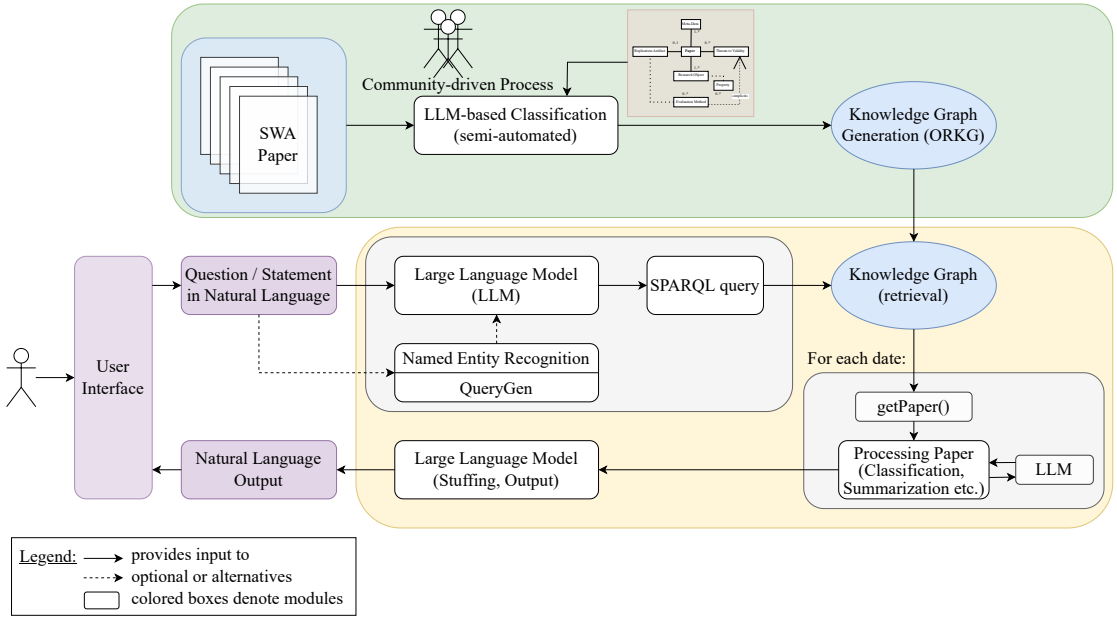


Figure 2.2.: The KARAGEN approach, as proposed by Kaplan et al. [53], consists of two parts for a LLM-based retrieval on the ORKG. The green box shows the generation and population of the ORKG, while the yellow box depicts the LLM-based retrieval.

complementary strengths of LLMs and KG. In this relationship, the KG provides a structured data foundation in which information is stored in an interconnected network of entities and their relationships. The benefit of the graph is the ability to semantically link information, thus preserving the context and relationships between pieces of knowledge. On the other hand, LLMs excel in interpreting and generating natural language, with their advantage being that users can formulate their queries in natural language instead of relying on keyword-based queries. [53]

The KARAGEN approach is illustrated in Figure 2.2 and is composed of two parts: 1) Knowledge Graph Generation and Population and 2) Knowledge Retrieval. The following explanations are based on the descriptions provided by Kaplan et al. [53].

Knowledge Graph Generation and Population This component is responsible for populating the ORKG with information through a semi-automated process, where users are supported by LLM-based classification. In the case of the ORKG, adding information about research papers is facilitated through templates. This means that either users or machines can fill in the required information into these templates, which are then stored in the graph as a contribution to the research paper.

Knowledge Retrieval This component handles the retrieval and processing of knowledge from the graph. Users can submit queries in natural language, which are then processed using an LLM. The retrieval process may involve several steps, such as generating graph queries by matching natural language elements to graph components. Optionally, a knowledge enhancement component can be incorporated to transform and generate information.

Finally, the system generates a response for the user. During this retrieval process, sources can be added to indicate the origin of the data and ensure traceability.

2.2. Knowledge Graph Question Answering (KGQA)

Question Answering over Knowledge Graphs (KGQA) presents a research area that focuses on empowering users to access information stored within a KG by formulating questions in natural language [15, 16, 17, 14, 18]. The primary objective of KGQA is to retrieve answers from a KG based on a question posed in natural language. This eliminates the necessity for users to write formal query languages such as SPARQL or possess intricate knowledge of the underlying structure and vocabulary of the KG [16, 17, 54].

Historically, the research objective of enabling automated systems to answer questions posed in natural language precedes the modern adoption of KGs. Early efforts in Question Answering (QA) explored systems designed to respond to questions by querying structured Knowledge Base (KB) or databases with early systems dating back to the 1970s and 1980s. A significant surge in interest in natural language question answering occurred with the establishment of a QA track at the Text Retrieval Conferences (TREC), beginning in 1999. [55]

Due to advancements in semantic web technologies and automated information processing, the creation of large volumes of structured information became feasible [17]. KGs emerged as a prime instance of storing structured data, which model facts about entities and their relationships, often in collections of triples [17, 56, 36]. Consequently, KGQA systems were developed to provide an intuitive natural language interface to query this structured knowledge contained in KGs. Over time, several terms have been used for this task, including Question Answering over Knowledge Bases (KBQA) [34] and Scholarly Question Answering (SQA) [54]. However, the term KB has often been used misleadingly in the literature as a synonym for KGs [34].

As KGQA research progressed, the focus from simple questions requiring the retrieval of single facts shifted to more complex ones requiring multi-hop reasoning or constraints [57, 15, 16]. Two primary research directions emerged for building KGQA systems: Methods based on *Information Retrieval (IR)* gather question-related information from the KG and leverage it to optimize the generation process. *SP-based methods* focus on the translation of the natural language question into a formal query or logical form that can be executed directly against the KG.

Neural networks became integral to various aspects of KGQA systems across both IR and SP paradigms [17, 36]. Employed neural network architectures include embedding models, Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Graph Neural Networks (GNNs) [16, 14, 36]. More recently, the advent of pre-trained LLMs has begun to impact research on KGQA. Although some research explores whether LLMs themselves can replace KGs, most research is directed at the unification of LLMs and KGs [14]. In current research, LLMs are integrated into KGQA systems in various ways. For

instance, they can act as entity/relation extractors to extract data from questions, are used in SP approaches to translate questions into formal queries, or are used in IR-based approaches to function as reasoners over retrieved KG subgraphs [14].

2.3. Large Language Models

Large Language Models (LLMs) are pre-trained models with hundreds of billions of parameters generated by excessive training of very large corpora of text [58]. These models are designed to estimate the probability distribution in a sequence of text, enabling them to perform a wide range of language-related tasks [59]. Recently, LLMs have demonstrated powerful capabilities in natural language understanding, expressiveness, general knowledge, and zero-shot generalization [7].

The architecture of most LLMs is derived from the *Transformer* design, which includes encoder and decoder modules empowered by a self-attention mechanism [14]. Based on their structure, LLMs can be broadly categorized into *encoder-only*, *decoder-only*, and *encoder-decoder* models. Notable examples of these architectures are Flan T5 [58] for the encoder-decoder framework, GPT-4 [60] for the decoder-only framework, and DeBERTa [61] for the encoder-only framework. In this thesis, we focus on the decoder-only framework, as many state-of-the-art LLMs follow this architecture [14].

2.3.1. Prompting

An important concept in leveraging LLMs is *prompting*, which involves conditioning the model on a natural language instruction to guide its generation for desired tasks without any further training or gradient updates to its parameters [62, 59]. Given this input prompt, the LLM then generates an answer in natural language. Popular prompting methods are *few-shot prompting*, which involves adding a few examples [63] or only adding instructions describing the task, which is referred to as *zero-shot prompting* [59]. Another popular prompting method is Chain-of-Thought (CoT), where a chain-of-thought demonstration is provided during prompting [64]. To maximize the effectiveness of LLMs, *prompt engineering* is a new field that focuses on the creation and refinement of prompts [14].

2.3.2. Limitations

Despite their impressive capabilities, LLMs face certain limitations. They are often described as black-box models, which means that they lack explainability and transparency, making it difficult for users to comprehend the reasoning behind their answers [7, 14]. Furthermore, LLMs are prone to generating factual errors or *hallucinations* in which the LLM provides answers that contradict existing sources or lack evidence to support statements, which is particularly prominent when handling queries beyond their training data or when requiring

current information [10, 7]. Moreover, LLMs may not perform as well on domain-specific tasks as on general ones due to the limited availability of domain-specific corpus [7].

2.3.3. Embedding Models

A challenge concerning natural language processing is the representation of words and their associated meanings. One paradigm to approach this challenge is the use of vector representations in a continuous space. The fundamental idea behind these representations is that words that appear in similar contexts tend to have similar meanings. Consequently, within such a vector space, the proximity between word representations encodes semantic and syntactic regularities. These dense vectors are typically referred to as *embeddings*. [15, 65, 66]

Early methods for learning representations of words, such as Word2Vec [66] and GloVe [65], train word embeddings based on the co-occurrence statistics of words within a large text corpus. This class of embeddings is termed *static* because the matrix produced during training is used without alteration. To address this limitation, *dynamic* word embeddings utilize the advances of the transformer model to compute different representations for the same word depending on its context within the given sentence. Consequently, transformer-based pre-trained language models such as BERT [67] have become relevant to solving the challenge of word representation. [15]

2.4. Graph Retrieval Augmented Generation (GraphRAG)

The paradigm of Retrieval-Augmented Generation (RAG) has gained significant traction to mitigate issues with LLMs, such as hallucination, lack of domain-specific knowledge, and outdated information. The concept behind RAG is to improve the outputs of an LLM by providing it with relevant information retrieved from an external knowledge source at the time of inference instead of relying on the potentially outdated or imprecise knowledge internalized during pre-training. [8, 68, 13, 7]

A typical RAG system involves a *retriever* component that obtains relevant information from a KB and a *generator* component that uses an LLM to generate a response based on the information retrieved and the question [68, 69].

Graph Retrieval Augmented Generation (GraphRAG) represents an instantiation of the RAG paradigm where instead of retrieving information from unstructured text documents, GraphRAG specifically targets and retrieves structured knowledge from graph databases such as KGs or text-attributed graphs [13]. Consequently, GraphRAG is positioned at the intersection of several key areas: traditional KGQA, the RAG paradigm, and the emerging field of unifying LLMs and KGs [14, 13].

A typical workflow of GraphRAG comprises three main stages [13]:

1. **Graph-based Indexing:** This constitutes the initial phase of GraphRAG, where a pre-existing graph database is selected or constructed to then establish indices upon it to later facilitate efficient retrieval.
2. **Graph-Guided Retrieval:** Given the provided input question, this phase aims to identify and extract a relevant subset of the graph data that is useful to provide an answer to the question.
3. **Graph-Enhanced Generation:** In this final stage, the question and the retrieved graph knowledge are used as input for an LLM to produce the natural language response.

2.5. Approximate Nearest Neighbor (ANN) Search

Approximate Nearest Neighbor (ANN) search is a technique used to find data points that are likely close or similar to a given query point in a high-dimensional vector space. Unlike Exact Nearest Neighbor (ENN) search, which guarantees finding the absolute closest data points, ANN search trades some accuracy for significantly improved efficiency and speed [70, 68]. The core idea behind ANN search is to preprocess the data into an indexing structure that tries to achieve a trade-off between search quality and search efficiency [68].

To understand the value of ANN search, it is helpful to first consider ENN search. Given a dataset of vectors $x_i, i = 1..N \subset \mathbb{R}^d$ and a query vector $q \in \mathbb{R}^d$, a typical ENN search involves finding the index n of the database vector x_n that minimizes the distance to q, typically using a distance metric such as the Euclidean Distance (L2). Other popular similarity metrics are cosine similarity or inner product similarity, for which the objective would be to maximize the score. [70]

A straightforward technique for ENN search is brute force, which involves calculating the distance between the query vector q and every single database vector x_i and then identifying one or more vectors with the smallest distance. The amount of vectors is often denoted by a k parameter. However, this process is slow on large datasets where ANN search comes in handy. Here, database vectors are preprocessed in a specialized indexing structure to allow for quickly narrowing down the search space at query time. [70, 68]

Various different ANN indexing techniques exist. *Inverted File System with Product Quantization (IVFPQ)* involves the partitioning of the vector space by applying clustering algorithms to then use fine-grained quantization to compress the vectors. *The Hierarchical Navigable Small World (HNSW)* is a technique that builds hierarchical graph structures where each node represents a vector. *Tree-based Indexing* organizes the vectors into hierarchical tree-like structures to separate the vector space. [68]

2.6. Evaluation Metrics

To evaluate GraphRAG approaches, the evaluation metrics can be broadly categorized into two types: *generation* and *retrieval* quality [13, 69]. Evaluating the generation quality is about the assessment of the generated answer, while the retrieval quality is about the of retrieved information and the coverage of the answer.

2.6.1. Evaluating the Retrieval Component

The evaluation of retrievers dates back to early information retrieval research. Conventional metrics typically compare the retrieved contexts with a set of *golden-labeled* contexts. These metrics fall into two types: *Rank-agnostic* or *Rank-aware*, depending on whether they consider the order in which the contexts are retrieved [69, 71]. In the following, we introduce commonly used metrics based on Yu et al. [69], Ibrahim et al. [72], and Hu and Zhou [73].

Rank-agnostic Metrics These metrics measure the quality of retrieval without considering the position of the item in the list of retrieved contexts.

- **Accuracy:** An assessment of the ratio of correctly retrieved contexts compared to the total number of retrieved contexts.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  True Positive (TP) refers to contexts that are relevant to the query and are accurately retrieved. True Negative (TN) refers to contexts that are irrelevant to the query and have not been retrieved. False Positive (FP) refers to contexts that are irrelevant to the query but have been incorrectly retrieved. False Negative (FN) refers to contexts that are relevant to the query but have not been retrieved.

- **Precision:** Measures the proportion of relevant contexts retrieved by the system to the total number of contexts retrieved by the system.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall:** Quantifies the proportion of relevant contexts that have been retrieved from the total number of relevant contexts for a given query, considering the top k results.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1:** An assessment which calculates the harmonic mean of precision and recall.

$$\text{F1} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Rank-aware Metrics These metrics additionally evaluate the order in which relevant items are presented. Typically, the higher the relevant item is placed in the list, the better the score.

- **Hits@k** A measure of the fraction of correct retrieved contexts that appear in the top k total of retrieved contexts.

$$\text{Hits@k} = \frac{H_k}{N_{query}}.$$

  In the above formula, H_k is the number of times a relevant context entry is in the top-k and N_{query} is the total amount of retrieved context.

- **Mean Reciprocal Rank (MRR)** Measures the average of the inverse ranks of the first relevant context retrieved by the system. This means that the metric focuses on how high the retriever ranks the first relevant context.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

  In the formula given above, $|Q|$ represents the total number of queries, and $rank_i$ denotes the rank position of the first relevant document for the i-th query.

- **Mean Average Precision (MAP)** Computes the average of the precision values at different cut-off points for each query. Consequently, this metric gives an aggregate view on how well the retriever ranks the relevant contexts.

$$MAP = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{\sum_{k=1}^{n} (P(k) \times rel(k))}{|relevantdocuments_q|}$$

  In the formula given above, $P(k)$ represents the precision at position k in the ranking list, and $rel(k)$ is an indicator function that is one if the document at rank k is relevant and zero otherwise. In addition, n denotes the total number of documents retrieved.

- **Exact Match (EM)** quantifies the proportion of retrieved contexts that exactly match the expected contexts.

$$ExactMatch = \frac{PEM}{N_{pred}}$$

  Where in the formula PEM is the proportion of the exact matches and (N_{pred} is the total number of expected contexts.

2.6.2. Evaluating the Generation Component

To assess the quality of the generated answer, often traditional metrics are borrowed from other natural language processing tasks like machine translation or summarization Yu et al. [69], Ibrahim et al. [72], and Alinejad, Kumar, and Vahdat [71, 71]. However, these metrics often fail to fully capture the performance within an LLM-based QA system [71]. Consequently, LLM models are employed as evaluative judges to assess the quality of generated answers [74].

Traditional Natural Language Processing Generation Metrics Common metrics that are applied to the evaluation in RAG are Yu et al. [69], Ibrahim et al. [72], and Alinejad, Kumar, and Vahdat [71, 71]:

- **ROUGE** Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is a set of metrics that evaluate the quality of the generated text by comparing it to the ground truth. There are multiple variants available: ROUGE-N calculates recall by comparing the presence of n-grams in the generated text and the ground truth. ROUGE-L uses the longest common subsequence to capture meaning in the generated text. ROUGE-W additionally uses weights for consecutive matches, distinguishing between spatially aligned and scattered matches. ROUGE-S incorporates skip-bigrams to balance flexibility and structure sensitivity. ROUGE-SU is a variant of ROUGE-S that uses skip-bigrams and unigrams to evaluate the quality of the generated text [75].
- **BLEU** The Bilingual Evaluation Understudy (BLEU) metric measures the quality of the generated text by comparing it with the ground truth by calculating the overlap of n-grams[76]. The Bilingual Evaluation Understudy (BLEU) score is calculated as:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

  In this formula, BP is the brevity penalty that penalizes short-generated translations, w_n is the weight for the n-gram, and p_n is the precision of the n-grams that match the reference.
- **BertScore** This metric uses contextual embeddings that, unlike n-gram-based metrics, capture the semantic meaning of the generated text. BertScore uses pre-trained transformer models like Bert to transform the generated text and the ground truth into embeddings. With the embeddings, the cosine similarity between the generated text and the ground truth is calculated, resulting in precision, recall, and f1 scores [77]

LLM as a Judge Metrics LLMs can be prompted with an evaluation scheme to assess the answers based on user-defined metrics. They have shown a great ability to capture semantic nuances and attend to variations in answers [71, 69]. Recently, a framework has been established focused on providing different LLM-based metrics [74]. Their framework is available online[3] from which the following metric explanations have been taken from:

[3] `https://github.com/explodinggradients/ragas` [last accessed on 26.03.2025]

- **Faithfulness** Measures the factual consistency of the generated answer against the given context, where the answer is regarded as faithful if all the claims made by the answer can be inferred from the given context.

$$\text{Faithfulness} = \frac{\text{Number of supported claims}}{\text{Total number of claims}}$$

- **Answer Relevancy** Focuses on how relevant the answer is to the question where higher scores indicate better alignment while lower scores are given if the answer is incomplete or includes redundant information. To calculate the score, an LLM generates N questions based on the generated answer and then calculates the cosine similarity between the actual question and the generated questions.

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^{N} \text{cosine similarity}(E_{g_i}, E_o)$$

  Where N denotes the number of generated questions, E_{g_i} the embedding of the i-th generated question, and E_o is the embedding of the actual question.

- **Factual Correctness** evaluates how well the generated answer aligns with a golden reference. The LLM decomposes both answers into individual claims and computes TP, FP, and FN as follows:

$$\begin{aligned} \text{True Positives (TP)} &= \text{Claims in the answer also found in the reference} \\ \text{False Positives (FP)} &= \text{Claims in the answer not found in the reference} \\ \text{False Negatives (FN)} &= \text{Claims in the reference not found in the answer} \end{aligned}$$

  Precision, recall, and F1 scores are then calculated using these values, as defined above.

2.6.3. Micro vs. Macro Averaging

When evaluating the performance of systems across a dataset of multiple questions, the overall performance needs to be assessed through aggregation. There exist two common types of aggregate scores: *micro-averaging* and *macro-averaging* [73]:

1. **Micro-Averaging** calculates the aggregation of metrics globally, for example, by counting the overall TP, FN, TN, and FP scores to calculate micro-Precision, micro-Recall, and micro-F1. This approach gives equal weight to each individual retrieved-context across all questions. Consequently, questions that involve a larger number of such individual contexts will have a proportionally greater influence on the overall micro-averaged score.
2. **Macro-Averaging** calculates each metric independently for each question and then takes the unweighted mean of the resulting scores. This essentially gives equal weight to each question, regardless of the number of contexts requested. It prevents the metric from being skewed by the performance of majority classes and ensures that the performance of minority classes is also represented.

The choice of the preferred averaging method depends on the evaluation goals. In LLM-based evaluations, *macro-averaging* is the preferred metric as observed by Hu and Zhou [73]. The authors suggest that this calculation is preferred as it ensures that all classes, regardless of size, are considered equally.

2.6.4. Sustainability Metrics

Kaplan et al. [78] highlight the importance of integrating sustainability metrics into the evaluation of natural language processing systems. They advocate for assessments that consider both performance and environmental aspects by tracking the energy consumption measured in kilowatt-hours (kWh) or megajoules (MJ) and the carbon emissions in CO_2. The authors propose several metrics for evaluating sustainability either by using energy consumption or carbon emissions:

- **Total Carbon Emission (CE):** Represents the absolute carbon footprint, measured in kilograms of CO_2 equivalents (kg CO_2e), resulting from the operation of the system during training or inference.
- **Relative Carbon Emission (CE_rel):** Links environmental impact directly to performance metrics, facilitating the assessment of carbon efficiency per unit of performance. The performance metric should be chosen based on the context.

$$\mathrm{CE}_{rel} = \frac{\mathrm{CE}}{\text{Performance Metric}}$$

- **Delta-based Carbon Emission (ΔCE):** To compare different systems performance improvements against carbon emissions. Using the lowest-performing system as a baseline, this metric reveals the environmental cost-effectiveness between two systems:

$$\Delta\mathrm{CE} = (\text{Performance Metric} - \text{Performance Metric}_{base}) \times \frac{\mathrm{CE}_{base}}{\mathrm{CE}}$$

- **Normalized Carbon Emission (nCE and $\mathbf{nCE_{rel}}$):** Normalizes the carbon emissions between the best and worst performing systems, providing a standardized scale (0 to 1) for easier comparison across different evaluations:

$$n(\mathrm{CE}) = 1 - \frac{\mathrm{CE} - \mathrm{CE}_{lowest}}{\mathrm{CE}_{highest} - \mathrm{CE}_{lowest}}$$

$$n(\mathrm{CE}_{rel}) = 1 - \frac{\mathrm{CE}_{rel} - \mathrm{CE}_{rel,lowest}}{\mathrm{CE}_{rel,highest} - \mathrm{CE}_{rel,lowest}}$$

2.7. Taxonomies

A taxonomy is understood as a classification system designed to organize objects or concepts [79, 80, 81, 82]. Although terms such as *classification*, *framework*, and *typology* are sometimes used interchangeably, the term *taxonomy* is used most frequently in the literature [81]. Consequently, for the remainder of this thesis, we will also use the term taxonomy.

Historically, taxonomies have been fundamental to research and practices across numerous disciplines, including natural sciences, social sciences, organizational science, strategic management, and information systems, as they are essential systems for structuring and understanding complex bodies of knowledge [81, 79]. The essence of a taxonomy is the classification process, a crucial cognitive task integral to conceptualization, language, mathematics, statistics, and overall data analysis [83, pp. 7, 11]. A *classification* is described as a process or system of organizing objects into groups or classes based on their similarity [81].

2.7.1. Formal Definition

Formally, a taxonomy T can be understood as a set of n dimensions $D_i (i = 1, \ldots, n)$, where each dimension consists of $k_i \geq 2$ characteristics $C_{ij} (j = 1, \ldots k_i)$ such that for any object of interest a classification based on C_{ij} for each D_i can be applied [81]. Although some taxonomies require the classification of each dimension to be mutually exclusive, we consider this not to be a mandatory characteristic.

2.7.2. Approaches to Taxonomy Development

Historically, the methodologies utilized in the creation of taxonomies have traditionally been classified into two primary categories [83, 81]:

- **Conceptual (Deductive):** In this method, the researcher suggests categories or types derived from established theories, concepts, or models, typically without depending on empirical data, though such data may be employed for validation purposes.
- **Empirical (Inductive):** Here, the process begins with empirical data about objects, and the classification is derived from this data, frequently using statistical techniques like cluster analysis.
- **Hybrid:** This approach combines conceptual and empirical elements and allows taxonomy developers to move back and forth between theoretical conceptualization and empirical observation.

However, often there is no systematic approach used for the creation of a taxonomy as it is created ad-hoc on the basis of intuition [81, 80, 79]. To provide researchers with a systematic framework that helps in the creation of taxonomies, several approaches have

been defined in the literature [81, 80, 79, 84]. One such approach has been proposed by Usman et al. [80], which is based on the taxonomy construction process from Bayona-Oré et al. [84]. This process involves five phases:

1. **Planning** Is about defining the context and initial setting of the taxonomy.
2. **Identification and Extraction** Is about the identification of terms that are relevant for the taxonomy with its subsequent extraction.
3. **Design and Construction** Is about analyzing the extracted terms to identify and describe dimensions, categories, and relationships. In addition, guidelines for the application of the taxonomy should be provided.
4. **Testing and Validation** Is about evaluating whether the taxonomy is useful to achieve the desired goals.
5. **Deployment** Is about the application of the taxonomy.

2.7.3. Evaluating a Taxonomy

Although many taxonomies are observed to be rarely evaluated and often developed in an ad-hoc manner [80, 79], structured methods for the evaluation exist. To validate taxonomies, Kaplan et al. [82] present a three-step evaluation method to evaluate the structure, applicability, and purpose.

First Step: Suitability The first step is to evaluate the suitability of the taxonomy structure by assessing the *generality*, *appropriateness*, and *orthogonality*. The generality is measured by *laconicity* and *lucidity*, while appropriateness is measured by *completeness* and *soundness*.

Given the classes $c \in C$ in the taxonomy C. Let $\mathcal{R}$ be a finite set of objects under study where each object under study $R \in \mathcal{R}$ has relevant terms $r \in R$.

- **Laconicity** assesses if terms in the objects under study map to at most one class, indicating the taxonomy is not too fine-grained:

$$\text{laconicity}(C, \mathcal{R}) = \frac{\sum_{R \in \mathcal{R}} \sum_{r \in R} \text{laconic}(C, R, r)}{\sum_{R \in \mathcal{R}} |R|} \in [0, 1]$$

  where $\text{laconic}(C, R, r) = 1$ if r maps to at most one c, and 0 otherwise.

- **Lucidity** assesses if each class maps to at most one relevant term, indicating the taxonomy is not too coarse-grained:

$$\text{lucidity}(C, \mathcal{R}) = \frac{\sum_{c \in C} (\min_{R \in \mathcal{R}} \text{lucid}(C, R, c))}{|C|} \in [0, 1]$$

  where $\text{lucid}(C, R, c) = 1$ if c maps to at most one r, and 0 otherwise.

- **Completeness** assesses if all relevant terms in the objects under study are covered by at least one class:

$$\text{completeness}(C, \mathcal{R}) = \frac{\sum_{R \in \mathcal{R}} \sum_{r \in R} \text{complete}(C, R, r)}{\sum_{R \in \mathcal{R}} |R|} \in [0, 1]$$

  where $\text{complete}(C, R, r) = 1$ if r maps to at least one c, and 0 otherwise.

- **Soundness** assesses if every class in the taxonomy maps to at least one relevant term:

$$\text{soundness}(C, \mathcal{R}) = \frac{\sum_{c \in C} (\max_{R \in \mathcal{R}} \text{sound}(C, R, c))}{|C|} \in [0, 1]$$

  where $\text{sound}(C, R, c) = 1$ if c maps to at least one r, and 0 otherwise.

- **Orthogonality Matrix** The orthogonality ensures that classes are distinct and non-overlapping. It is assessed by using a self-referencing orthogonality matrix where entries indicate dependencies between classes and fewer dependencies indicate a better overall orthogonality.

Second Step: Applicability The second step is to define whether the taxonomy can be used consistently and effectively. When possible, this should be evaluated by different users, often through user studies. Here, the *reliability* of the user results can be evaluated by using inter-annotator agreement metrics like Krippendorff's α. Furthermore, *correctness* can be employed to assess how accurately users apply the taxonomy compared to a gold standard, using metrics like precision, recall, and F1. Finally, *ease-of-use* is about how easily users understand and apply the taxonomy, evaluated via questionnaires.

Third Step: Purpose The third and final step of evaluation is the assessment of purpose. This involves the evaluation of the *relevance*, *novelty*, and *significance*. Here, the novelty measures the degree of innovation and adaptation compared to existing taxonomies and is assessed by employing the metrics *innovation* and *adaption*. The significance evaluates if the taxonomy offers a more detailed categorization than predecessors and is assessed through a *classification delta*.

- **Relevance** assesses whether each category and class contributes meaningfully to the intended purpose. It can be quantified by argumentatively assessing the relevance and then calculating the fraction of relevant classes and categories.

- **Innovation** assesses the proportion of classes or categories in the evaluated taxonomy that are entirely new compared to a set of previous taxonomies $\mathcal{T}$:

$$\text{innovation}(C, \mathcal{T}) = \frac{\sum_{c \in C} \min_{T \in \mathcal{T}} \text{new}(C, T, c)}{|C|} \in [0, 1]$$

  where $\text{new}(C, T, c) = 1$ if c is not equal to or adapted from any $d \in T$, and 0 otherwise.

- **Adaption** assesses the proportion of classes or categories in the evaluated taxonomy that have been adapted from classes or categories in previous taxonomies $\mathcal{T}$:

$$\text{adaptation}(C, \mathcal{T}) = \frac{\sum_{c \in C} \max_{T \in \mathcal{T}} \text{adapted}(C, T, c)}{|C|} \in [0, 1]$$

  where $\text{adapted}(C, T, c) = 1$ if $c \simeq d$ for any $d \in T$, and 0 otherwise.

- **Classification Delta** assesses whether the evaluated taxonomy C provides a more detailed categorization for a set of objects $\mathcal{R}$ compared to the most detailed previous taxonomy in $\mathcal{T}$:

$$\text{classification\_delta}(C, \mathcal{T}, \mathcal{R}) = \frac{|\sim_C| - (\max_{T \in \mathcal{T}} |\sim_T|)}{|\mathcal{R}|} \in [-1, 1]$$

3. Related Work

This chapter outlines related work associated with our contributions. As discussed in the introduction, this thesis centers on the subject of KGQA in the context of the scholarly literature task. We propose a novel approach, a taxonomy, and a dataset focusing on this topic. Accordingly, the discussion of related work is organized around these three areas.

In the following sections, we first summarize current work that proposes KGQA retrieval approaches, which we see related to our proposed HubLink approach. Then, we focus on the construction of taxonomies as it has been proposed by previous literature and existing taxonomies or classifications for KGQA. Last, we investigate datasets for benchmarking on KGQA and the construction of such datasets. Although this thesis aligns with these research domains, it introduces distinct advancements that contribute to progress in each respective area.

3.1. Using Large Language Models for KGQA

In the following, we examine existing KGQA retrieval strategies that use an LLM and are relevant to our proposed HubLink approach (**C1**) across four categories. First, we introduce methods that specifically target the scholarly domain. Next, we focus on KGQA approaches that utilize an LLM to conduct stepwise reasoning on the graph. Then, we review techniques centered on dynamic subgraph construction to conduct inference for KGQA. Last, we conclude with methods based on converting graph knowledge into a vector space to perform inference using distance measures.

3.1.1. Approaches for QA on Scholarly Knowledge Graphs

Recent efforts in scholarly QA have explored the integration of LLMs with RKGs, primarily through *semantic parsing* methods. These approaches translate natural language questions into formal queries, such as SPARQL, by identifying relevant entities and relations in the underlying KG [85].

JarvisQA [28] leverages tabular data converted into triples and uses a language model to textualize these triples. The final answer is then retrieved using a BERT-based model [67]. However, this system operates on structured tabular data provided by the ORKG rather than performing direct reasoning over an RKG. Similarly, DBLP-QuAD [24] fine-tunes a T5 model [86] to translate natural language questions into SPARQL queries. However,

this method depends on large annotated datasets and effective entity linking, which are often unavailable for newly constructed scholarly KGs. Another approach, KGMistral [87], utilizes a RAG framework where pre-prepared SPARQL query templates are filled to directly retrieve relevant triples from the KG. These triples are then converted into natural language and provided as context for the LLM. Similarly, Taffa and Usbeck [25] propose a retrieval-based strategy that reuses the pre-prepared SPARQL query associated with the most similar question in the training set. Although efficient in controlled settings, this reuse of existing templates struggles to generalize beyond the training distribution.

The SciQA dataset has been central to the evaluation of most of these methods, where they have shown strong results. However, these results can be misleading because the questions in the dataset have been automatically generated from seed questions, which an LLM might be able to identify inherent patterns from [26]. Moreover, many of the test questions in the dataset share schema elements with training data, reducing the challenge of generalization [27]. Finally, this benchmark does not take into account that semantic parsing methods tend to degrade when applied to larger, dynamic KGs, where unseen schema components and entities are common [29]. Furthermore, their reliance on task-specific training examples limits adaptability and scalability.

In response, prompt-based and schema-agnostic semantic parsing methods have been introduced. Lehmann et al. [26] compare fine-tuned and prompting-based approaches on SciQA, finding that zero-shot prompting has a poor performance but that few-shot prompting can rival fine-tuning in accuracy while offering better adaptability. Still, using few-shot prompting requires examples that do not translate well to unseen data. Next, Jiang, Yan, and Usbeck [27] propose a two-stage prompting framework that incorporates ontology information into both structural and content-level prompts. Although this strategy achieves strong performance, its effectiveness diminishes with larger ontologies, posing scalability challenges.

In contrast to these approaches, the proposed HubLink retrieval approach is not semantic parsing based and does not require annotated training data or prior schema knowledge. Instead, it embeds subgraphs rooted in specific nodes and generates text representations of paths using a general-purpose embedding model. This allows the method to dynamically adapt to evolving schema, making it more robust and scalable for real-world applications.

3.1.2. LLM-Guided Stepwise KGQA Approaches

Many current training-free KGQA approaches proposed in the literature are stepwise reasoning approaches. These methods typically decompose the complex reasoning task into smaller, manageable steps, iteratively querying the KG and using the LLM to guide the process towards an answer. Similarly to our HubLink approach, many of these methods utilize a pre-trained LLM without requiring task-specific fine-tuning. However, they fundamentally differ in how they interact with and structure the KG information during reasoning.

A prominent strategy involves employing the LLM as an active agent or controller within the reasoning process. For instance, Think-on-Graph (ToG) [88] actively involves the LLM in

multi-hop reasoning by using a beam search strategy where the LLM evaluates and prunes potential reasoning paths. Knowledge Solver (KSL) [89] frames KG traversal as a decision-making process within a dialogue-like prompt structure, where the LLM selects subsequent entities based on relation prompts. Sun et al. [90] propose an Observation-Driven Agent (ODA), which operates in a cycle of observation, action, and reflection. In this cycle, a subgraph is expanded based on similarity, an LLM selects between further exploration, path discovery, or answering, and the internal memory is updated to guide future steps. GRAPH-COT [91] uses the LLM to hypothesize required information, issue-specific graph function calls such as node retrieval or neighbor inspection, and integrate the results in a final answer. StructGPT [92] provides a general framework using an Iterative Reading-then-Reasoning process, linearizing information extracted from structured sources like KGs or tables via specialized interfaces that the LLM uses. FiDeLiS [93] combines a Path-RAG component for retrieving relevant entities and relations with a deductive verification beam search module where the LLM iteratively assembles and validates multi-hop paths.

There are also approaches that extend the retrieval to further enhance the context available to the LLM. For example, hybrid strategies such as ToG-2 [94] integrate both structured KG exploration and unstructured text retrieval during the inference to iteratively refine the search using LLM guidance. Other approaches address KG incompleteness. For example Generate-on-Graph (GoG) [95] allows the LLM to supplement missing facts by generating plausible triples based on its internal knowledge alongside retrieved KG information.

Although the traversal strategy of HubLink incorporates an iterative process for subsequent retrievals if initial information is insufficient, its core mechanism diverges significantly from the stepwise reasoning methods mentioned above. The approaches discussed above primarily focus on dynamic, on-the-fly traversal and reasoning, where the LLM often guides the exploration entity by entity. In contrast, HubLink involves retrieving precomputed and embedded subgraph structures referred to as *hubs* rooted at specific node types. The reasoning and inference process operates predominantly on these retrieved structured hub representations rather than solely on the sequence of entities traversed during a dynamic search. This pre-structuring and retrieval of meaningful subgraphs is the central difference to other stepwise approaches. We argue that by using this strategy, the transitive relations of entities not yet reached can be incorporated into the reasoning, a problem that current stepwise reasoning approaches struggle with, as outlined in Section 10.3 of the thesis.

3.1.3. KGQA using Contextual Subgraph Construction

Another related line of research focuses on constructing subgraphs dynamically during the retrieval process to provide contextual information for KGQA, leveraging LLMs without additional training or fine-tuning. These methods typically identify relevant entities in the question and explore the KG to build a localized graph structure relevant to the query.

For example, Mindmap [96] uses prompts to guide an LLM in constructing path-based and neighbor-based evidence subgraphs, which are then aggregated into a reasoning graph used

for answer generation in order to preserve the structure of the graph during inference. KG-GPT [97] first decomposes the question into potential triples, retrieves candidate relations for identified entities, and builds an evidence subgraph from these components before having the LLM reason over the linearized result. Reasoning on Efficient Knowledge Paths (RoK) [98] uses a chain-of-thought approach to extract key entities, generates multi-hop reasoning paths between them, ranks these paths using PageRank, and forwards the resulting subgraph to the LLM.

Although these approaches utilize subgraphs as contexts for the LLM, they differ fundamentally from HubLink. The core distinction lies in both when and how the subgraphs are created. The methods described above construct subgraphs dynamically at query time, often starting from entities mentioned in the question and expanding based on neighborhood proximity, path generation, or prompt-guided assembly. In contrast, HubLink employs an offline indexing process where subgraphs or so-called *hubs* are precomputed based on predefined node types acting as semantic anchors. These hubs are designed to represent coherent semantic units, capturing meaningful relationships extending from the root node. HubLink retrieves these entire pre-constructed hubs rather than constructing subgraphs dynamically. This precomputation allows for faster retrieval and ensures that the retrieved context aligns with predefined semantic structures. Dynamic methods, on the other hand, build context based on immediate query features and graph topology without necessarily enforcing a specific semantic coherence or tracking information origin in the same way the hub definition of HubLink allows.

3.1.4. Utilizing Vector Representations for KGQA

Another direction of KGQA is the use of dense vector representations to capture knowledge and facilitate retrieval. These approaches vary in what they embed and whether they require dedicated training.

One category involves directly embedding the KG, using pre-trained models without KG-specific training. Direct Fact Retrieval (DiFaR) [99] exemplifies this by encoding individual KG triples into dense vectors during an offline phase. At query time, relevant triples are retrieved via nearest-neighbor search between the question embedding and the triple embeddings and then provided as context to an LLM. However, this approach primarily considers isolated triples, potentially missing the broader context captured in multi-hop paths or larger graph structures. HubLink does not embed isolated triples. Instead, it focuses on precomputed subgraph structures and embeds rich textual descriptions derived from the paths, triples, and entities within these hubs. This allows capturing more complex relational context than individual triples.

A different category involves training dedicated Knowledge Graph Embedding (KGE) models [100, 14]. These methods learn vector representations for entities and relations by training specific models on the structure of the KG. For example, Pretrain-KGE [101] integrates pre-trained language models like BERT and fine-tune them on KG triples. KEPLER [102] employs a dual-objective training combining KGE and masked language modeling. Nayyeri et al.

[103] use specialized algebraic embeddings like quaternion models in their training. These approaches aim to capture implicit semantic relationships within the KG but necessitate a training phase, often requiring significant graph data and computational resources. Crucially, HubLink leverages general-purpose pre-trained embedding models to encode these textual descriptions, requiring no dedicated KGE training or fine-tuning. This design choice makes HubLink easily applicable to different KGs without the need for training data or processes while still aiming to provide a semantically rich structured context for downstream LLM reasoning.

3.2. Taxonomy Development and Question Classification

As part of this thesis, we provide a taxonomy for the classification of characteristics of a KGQA system on the literature search task (**C2**). To support the development of this taxonomy, we also provide a systematic construction approach (**C2.1**). In the following, we first review the evolution of systematic approaches for taxonomy construction and evaluation. This involves discussing established methods from Information System (IS) and Software Engineering (SE), their practical application challenges, and identified limitations with respect to operationalization and systematic adaption of existing classifications. Then, we present an overview of the landscape in question classification with a particular focus on the application within QA systems and KGQA benchmarks. We highlight existing practices but also identify critical gaps concerning the need for a comprehensive and generalized taxonomy.

3.2.1. Systematic Approaches to Taxonomy Construction and Evaluation

Taxonomies serve as fundamental artifacts across numerous disciplines, including IS and SE. They provide essential structure for complex bodies of knowledge to facilitate understanding and enable research. Historically, taxonomies in both IS and SE have not been developed following any systematic approach [80, 81]. In response to this deficit, Nickerson, Varshney, and Muntermann [81] proposed a systematic method for taxonomy development, which is grounded in taxonomy literature from disciplines such as biology and social sciences. The approach outlines an organized sequence of steps, which involves identifying the meta-characteristic and ending condition criteria, followed by repeated cycles of empirical-to-conceptual or conceptual-to-empirical analysis depending on available data and the expertise of the researcher. This method is intended to steer researchers away from ad-hoc methods toward a more intentional and systematic approach. However, despite being widely referenced in the IS taxonomy literature since its publication, subsequent analysis by Kundisch et al. [79] revealed that the adoption of the Nickerson, Varshney, and Muntermann [81] method in practice has been inconsistent and often lacked transparency, which suggests challenges in the practical operationalization of the method. Consequently, Kundisch et al. [79] adapt the original taxonomy construction approach to directly address the observed

issue of inconsistent adoption, lack of transparency, and the infrequent evaluation of taxonomies.

Parallel efforts to provide systematic guidance can be observed in the SE domain. One of the first studies to describe a systematic approach to taxonomy development in SE is provided by Bayona-Oré et al. [84]. They present a method for creating taxonomies based on a systematic review and industry experience. Their approach consists of several phases, such as planning, identification and extraction, design and construction, testing and validation, and deployment. The approach was later revisited by Usman et al. [80], who identified limitations, including the inclusion of activities outside the scope of taxonomy design, its basis in the limited literature, and a lack of explicit guidance for crucial methodological decisions. In response, they proposed a revised method that reduces the number of phases and focuses only on essential activities while also introducing new ones to support more thorough and flexible taxonomy development.

Although significant advances have been made in providing systematic construction methods, as illustrated by the above methods [79, 81, 84, 80], the operationalization of these processes still presents challenges. The noted inconsistencies and calls for more implementation guidance [79] suggest that researchers may require more detailed, step-by-step, and potentially tool-supported guidance to effectively apply these systematic approaches. Furthermore, while methods for developing new taxonomies exist, the literature indicates that taxonomies are rarely revisited, revised, or extended [80]. This infrequent revision suggests that researchers tend to build new taxonomies from scratch or without clear reference to existing classifications rather than systematically synthesizing, adapting, and refining knowledge present in the existing literature. Although Kundisch et al. [79] recommend accounting for and referring to existing taxonomies, current methods primarily focus on the construction of new taxonomies. They lack a specific and operational process for systematically synthesizing and adapting existing taxonomies. This identified gap is addressed by our Contribution **C2.1**. Our systematic taxonomy construction approach provides a systematic process for initializing taxonomies based on existing literature, incrementally refining them using quantitative metrics (from Kaplan et al. [82]), and offering tool support to aid researchers in execution.

3.2.2. Question Classification for Scholarly QA on Knowledge Graphs

QA systems are typically assessed using benchmarks that include questions asked to such systems. To characterize these questions and understand which specific types of questions are used during evaluation, a taxonomy is helpful that provides guidelines for the classification of such questions. In the past, several works have utilized this approach to ensure diversity when developing datasets for KGQA. LC-QuAD 2.0 [31] incorporates 10 types of questions, such as single or multi fact, boolean, counting, or ranking. Furthermore, specifically targeting scholarly graphs, the dataset SciQA [104] includes various types of questions, including complex, factoid, and non-factoid questions, addressing aspects like superlatives, negation, counting, or ranking. SciQA also provides statistics on the distribution of questions across categories such as query shapes, query classes, query types,

query components, and triple patterns. Another notable dataset in the scholarly domain that also uses a taxonomy for its dataset design is DBLP-QuAD [24]. The authors explicitly categorize the questions into 10 different types such as single or multi fact, boolean, union, or disambiguation. Moreover, Jaradeh, Stocker, and Auer [28] in their work use types such as aggregation, ask, or listing to guide their construction of questions.

Beyond using question types for the development of benchmarks targeting KGs, the literature also presents classifications of questions in other contexts. Li and Roth [32] propose a two-layered taxonomy for question classification in text-based QA. Their classification is based on expected answer type semantics with six coarse classes, such as abbreviation, entity, or description, and 50 fine-grained classes. Moldovan et al. [105] provide an extensive taxonomy for open-domain QA systems, differentiating questions according to question classes, subclasses, answer types, and focus. Furthermore, Dillon [106] provides a theoretical discussion and philosophical perspective on the classification of research questions. This approach is based on the kinds of knowledge a question entails, proposing a categorical scheme such as character descriptions, questions of identification, and conditional questions. On a similar note, Thuan, Drechsler, and Antunes [107] examine the construction and formulation of research questions specifically in Design Science Research (DSR). They propose a taxonomy of question classification, including the types of problem-solving, gap-spotting, and problematization.

While these KGQA datasets [31, 104, 24, 28] provide crucial benchmarks for evaluating QA system capabilities across diverse question complexities, they primarily offer classifications tailored to their respective datasets, underlying knowledge graphs, or generation methodologies. None of these works specifically focuses on taxonomy creation itself. Instead, they present question types primarily as a secondary component that supports dataset construction. Consequently, these works do not present or refer to a comprehensive, generalized taxonomy for scholarly KGQA specifically designed for the literature search task. While other classification schemes exist [32, 105, 106, 107], they are not specifically designed for the nuances of scholarly KGQA. Therefore, an overarching taxonomy for scholarly KGQA in the literature search context is still a missing element in the current research landscape. Our Contribution **C2** addresses this gap by synthesizing existing classifications from the literature to form a comprehensive taxonomy.

3.3. Benchmarking KGQA Systems

To evaluate the performance of KGQA systems, numerous datasets have been introduced. A survey on KGQA methods and their benchmarks shows that the majority of current research is focused on the open domain [13]. Most of these benchmarks target general-purpose encyclopedic KGs such as Freebase [108], DBpedia [109], or Wikidata [110]. Notably, there is an underrepresentation of QA datasets targeting the academic domain. This domain presents unique challenges, as scholarly articles can be difficult to interpret, even for human experts [30].

In this section, we introduce related work that is relevant to our third Contribution **C3**, the KGQA dataset, and the construction approach that we applied. We first introduce KGQA datasets that target the open domain. Then, we focus on KGQA datasets specifically for the scholarly domain. Last, we introduce dataset construction methodologies related to our approach.

3.3.1. Datasets Targeting Open-Domain Knowledge Graphs

Early KGQA datasets focused on the Freebase graph, which was a large open-domain graph hosted by Google [111]. Free917 [112], for example, includes 917 questions formulated by two domain experts and manually annotated with formal queries, specifically targeting semantic parsing benchmarks. Additionally, WebQuestions [111] consists of 5,810 factoid question-answer pairs targeting Freebase data. The dataset is relatively simple in complexity, as many questions involve only a single class, property, or instance. Later, it was extended to create WebQuestionsSP [113] for semantic parsing benchmarking, associating SPARQL queries with 4,737 questions. A larger dataset focused on Freebase is SimpleQuestions [114], which consists of 108,442 factoid questions paired with answers. Due to its popularity, the dataset was later converted to SimpleQuestionsWikidata by Diefenbach et al. [115], who translated the question and answer pairs to the Wikidata graph.

The datasets mentioned above primarily feature simple factoid questions. Other datasets were developed to incorporate more complex question types. ComplexWebQuestions [116] is one of the first datasets featuring questions with superlatives and comparatives. It was created based on the WebQuestionsSP dataset and therefore also targets the Freebase graph. Similar complexity is exhibited by the Large-Scale Complex Question Answering Dataset (LC-QuAD) [117]. It comprises over 5,000 questions coupled with SPARQL queries necessary to obtain the answers from the DBpedia graph. The dataset was further developed by the authors, who released a second version: LC-QuAD 2.0 [31]. This updated dataset is larger than its predecessor, containing more than 30,000 questions that span both the Wikidata and DBpedia graphs. In addition, it includes a wider variety of question types and multiple paraphrases for each question. The Question Answering over Linked Data (QALD) challenges[1] aim to advance natural language interfaces for querying knowledge graphs like DBpedia and Wikidata. The most recent benchmark, QALD-10 [118], was published in 2022 and comprises 394 manually created questions derived from Wikidata, covering different levels of complexity. Each question is annotated with a manually defined SPARQL query and its corresponding output. In addition, the dataset contains questions in various languages, including English, German, Russian, and Chinese. The dataset features simple factual, comparative, superlative, count-based, and temporal questions.

In particular, all KGQA datasets in this section are mainly encyclopedic knowledge graphs focusing on the open domain. Our dataset specifically targets the *scholarly literature search* task in the SWA domain. The dataset contains a classification of questions into eight different

[1] `http://qald.aksw.org/` [last accessed 23.09.2024]

retrieval operations and six different use cases specifically for the literature search. This makes it possible to find out how well KGQA approaches work, especially with this task.

3.3.2. Datasets Targeting the Scholarly Domain

While most KGQA datasets target open-domain knowledge bases, efforts have emerged to construct benchmarks specifically for the scholarly domain. Several initiatives have targeted the ORKG as the underlying knowledge base. ORKG-QA [28] represents one of the first datasets built upon the ORKG. The dataset comprises 80 question-answer pairs based on 13 tables extracted from the ORKG, representing structured comparisons of over 100 academic publications. Karras et al. [37] provide a set of 77 competency questions to evaluate empirical research targeting the SWA domain. However, unlike our proposed dataset, these works do not explicitly categorize questions based on specific information retrieval operations relevant to literature search.

Recent scholarly datasets feature larger scale and greater complexity and include retrieval operation-related types. SciQA [104] provides 2,465 questions in the scholarly domain, with the ORKG as the underlying knowledge base. These questions are predominantly complex, requiring reasoning over multiple publications. Each question was generated with a template that corresponds to a typical scholarly inquiry to ensure its relevance. A larger dataset, DBLP-QuAD [24], consists of over 10,000 question-SPARQL pairs designed for QA over the DBLP computer science bibliography graph. The dataset comprises various question types, each presenting different levels of complexity for the retrieval process. Although these datasets address complex questions within the academic domain and also provide retrieval operation types similar to our KGQA dataset, the primary distinguishing feature of our dataset is its explicit focus on the literature search process. This is achieved by structuring the questions according to six specific literature search use cases, each incorporating relevant metadata and content constraints, allowing a targeted evaluation of KGQA systems for these practical scenarios.

3.3.3. Methodologies for Question Answering Dataset Construction

Several strategies have been developed for generating KGQA datasets. The most straightforward way is to handcraft the datasets. For example, in ORKG-QA [28], the authors collected 13 tables from the ORKG, representing structured comparisons, from which they manually formulated 80 natural language questions covering diverse types, including direct, aggregation, and relational queries. Additionally, Karras et al. [37] manually developed 77 competency questions for the ORKG, specifically targeting empirical research in software and requirements engineering. Similarly, Free917 [112] was created by two domain experts who manually generated 917 natural language questions accompanied by formal queries.

Another category of approaches employs crowd-sourcing techniques. Although this method typically does not involve domain experts for question creation, outsourcing allows for scaling datasets to larger sizes. For example, WebQuestions [111] was created by obtaining

100,000 candidate questions from the Google Suggest API and forwarding a random selection to Amazon Mechanical Turk. The human workers on the platform were assigned to annotate answerable questions using Freebase. The construction of SimpleQuestions [114] followed a similar crowd-sourcing process. First, a set of RDF triples was selected from Freebase. Then, the annotators were asked to write a natural language question for each triple in which the object serves as the answer, incorporating the subject and predicate. Amazon Mechanical Turk workers were also employed for the construction of ComplexWebQuestions [116]. In this case, their task was to paraphrase existing questions and to identify corresponding answers in Freebase.

A different technique of constructing KGQA datasets involves the instantiation of SPARQL templates. For example, SciQA [104] starts with a small set of manually authored question-SPARQL pairs over the ORKG. SPARQL templates are prepared, and an LLM is subsequently used to populate these templates automatically with entities and predicates, generating several thousand additional questions. Similarly, DBLP-QuAD [24] originates from 98 manually defined SPARQL and natural-language query templates. By sampling two-hop subgraphs from the DBLP graph and augmenting them through paraphrasing, the authors synthesized 10,000 question–query pairs. In a similar manner, LC-QuAD [117], which uses the DBpedia graph, employs manually defined SPARQL query templates parameterized by seed entities and a predicate whitelist. These templates were instantiated using two-hop subgraphs to produce the dataset.

Although our method also involves manually generated templates, our templates are fundamentally different. They consist of questions in natural language that contain placeholders for missing constraints rather than being based on SPARQL query structures. We provide these natural language templates to an LLM tasked with generating analogous questions populated with relevant entities and relations derived from the underlying knowledge graph data. A potential limitation of strictly template-based SPARQL generation is that it can constrain the resulting questions to the explicit entities and relations defined in the templates, potentially overlooking semantically related concepts. In contrast, our approach leverages dense vector representations of entities and relations. This allows for the generation of more complex and nuanced questions, including those involving entities that are semantically related rather than just structurally connected according to predefined templates.

However, employing embedding-based techniques for generating questions is not a novel concept. Another research direction leverages RAG with LLMs to index the context and generate questions from it. For example, [119] employs a RAG approach to specifically generate competency questions during ontology engineering. In their method, relevant text chunks from a corpus of scientific papers are retrieved based on embedding similarity and provided as context to an LLM to formulate questions. Although this technique uses embeddings for retrieval, its primary knowledge source consists of external documents rather than a KG.

4. HubLink: KGQA by Graph Decomposition

This chapter introduces HubLink, a novel approach for KGQA, representing the primary Contribution **C1** of this thesis. HubLink is specifically designed as a schema-agnostic and training-free method. Its core principle involves subdividing the graph into distinct subgraph structures, termed *Hubs*, during an indexing phase. These Hubs facilitate source-aware information retrieval at query time, enabling the identification of provenance for the retrieved information. This characteristic is particularly valuable within scholarly literature search contexts, where traceability of findings is essential.

The structure of this chapter is organized to systematically present the HubLink framework. It begins with a high-level overview of the approach in Section 4.1. Following this, Section 4.2 establishes the necessary formal definitions and concepts.

The subsequent sections provide a detailed explanation of the HubLink algorithm. Section 4.3 introduces the core data models utilized. The process of identifying the root entities of the hub within the graph is presented in Section 4.4. Section 4.5 then explains the comprehensive indexing procedure, which includes the construction and storage of Hubs. Operations specific to the vector store employed for indexing and retrieval are described in Section 4.6. Section 4.7 elaborates on the two distinct retrieval strategies implemented within HubLink. The algorithmic description concludes in Section 4.8, which details the answer generation and source-linking mechanisms.

After presenting the algorithm, the chapter discusses broader aspects of the HubLink framework. Section 4.9 examines the key design decisions made during development and provides their rationale. Section 4.10 evaluates the applicability of HubLink to domains beyond scientific literature and assesses its scalability potential for large graphs. Considerations for maintaining index updates over time are discussed in Section 4.11. Finally, an analysis of the limitations of the approach is provided in Section 4.12.

4.1. Overview

This section provides an overview of our proposed HubLink approach. The purpose of this section is to illustrate the overall processes involved before diving deeper into the details in subsequent sections.

HubLink is fundamentally a GraphRAG technique, which is characterized by its three graph-based stages: indexing, retrieval, and generation. In the following, Section 4.1.1 begins with the indexing process, a required preprocessing step that needs to be performed

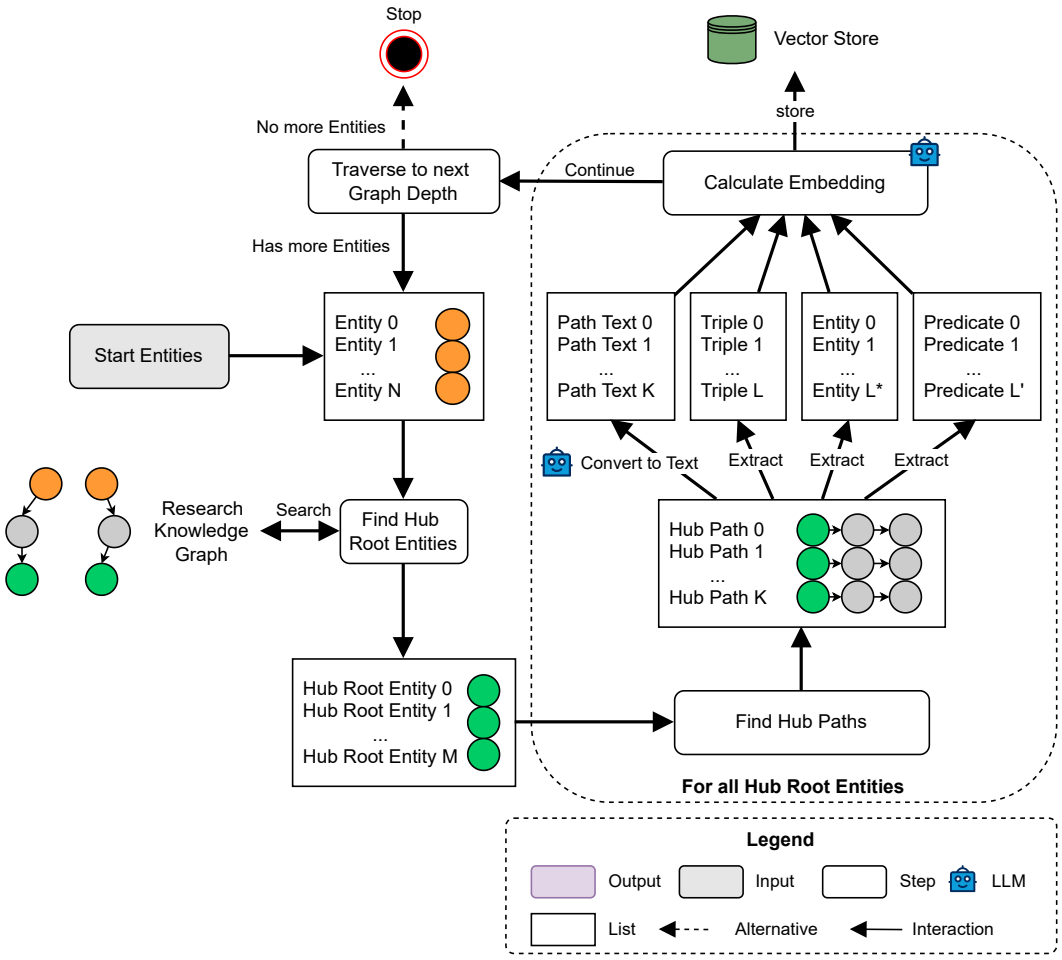


Figure 4.1.: An overview of the indexing process of HubLink showcasing how data is extracted from the KG and stored in a vector store.

before retrieval to decompose the graph into subgraph structures referred to as *hubs*. After indexing, the retrieval and generation process will be presented in Section 4.1.2. Here, the HubLink retriever utilizes one of two different strategies to retrieve relevant data from the index.

4.1.1. Indexing

The indexing process is illustrated in Figure 4.1. It begins with a set of start entities from the graph that serve as initial points for indexing. Using these entities, a search is performed with the goal of finding root entities of hubs. These are specific entities in the graph from which each hub is built. To find these roots, each possible path is traversed until either the end of the graph or until an entity in the graph is reached that is classified as the root of a hub. These identified entities are referred to as `HubRoot` objects, which are stored in a list to be processed sequentially.

For each identified `HubRoot` entity, the algorithm continues by finding and building the so-called `HubPaths`. These paths start from the `HubRoot` entity and lead to the end entities of a hub, which are either leaf nodes in the graph or other entities classified as roots of a hub. Both `HubPaths` and `HubRoot` form a `Hub` structure, which act as the central elements during retrieval.

Once the `HubPaths` of a hub are found, each path undergoes further processing before it can be indexed. First, the entire path is passed to an LLM to create a textual description of the path. In addition, an extraction of the triples, entities, and predicates from the path is performed. These pieces of information, the textual description, the triples, the entities, and the predicates, are then mapped into a low-dimensional vector space using a pre-trained embedding model. After this transformation, these vectors are stored in a vector store, which enables ANN search for fast access during the later retrieval stage. Furthermore, additional metadata are attached to each vector. This includes the root identifier of the hub for the identification of the hub to which the vector belongs and the description of the path that was generated previously by an LLM. This metadata is later required for the retrieval phase of the algorithm.

Once these hubs are indexed, the procedure repeats from the nodes in the graph that form the endpoints of each `HubPath`, provided they have not yet reached a leaf node. The indexing process stops until either the maximum number of traversal levels is reached or no new hubs are found. When the graph is updated, this update needs to be reflected in the index, which we discuss in Section 4.11.

4.1.2. Retrieval and Generation

The HubLink retrieval and generation process is illustrated in Figure 4.2. The retriever offers two strategies for extracting relevant contexts from the KG. The first strategy, *Direct Retrieval*, performs a global search on the whole index without requiring access to the graph during retrieval. The second strategy, named *Graph Traversal Retrieval*, performs a subgraph search that begins at a chosen entry point to explore the graph for the identification of relevant hubs to generate answers. The selection of the strategy involves balancing runtime and accuracy, a topic discussed later in this chapter. Both strategies differ only in certain parts of the retrieval algorithm, which we highlight in Figure 4.2 by distinguishing the paths with different colors.

The retrieval begins with an extraction of components from the question. Here, an LLM is queried to extract the most relevant parts of the question as a list. This list of components and the question itself are then transformed into vectors using the same pre-trained embedding model used in the indexing phase. Now, depending on the strategy applied, the following steps differ:

Graph Traversal Strategy: In addition to the question itself, this strategy requires as input a point of entry into the graph that is referred to as *Topic Entity*. The algorithm explores all paths starting at the topic entity to find `HubRoot` entities. After collecting these entities, an

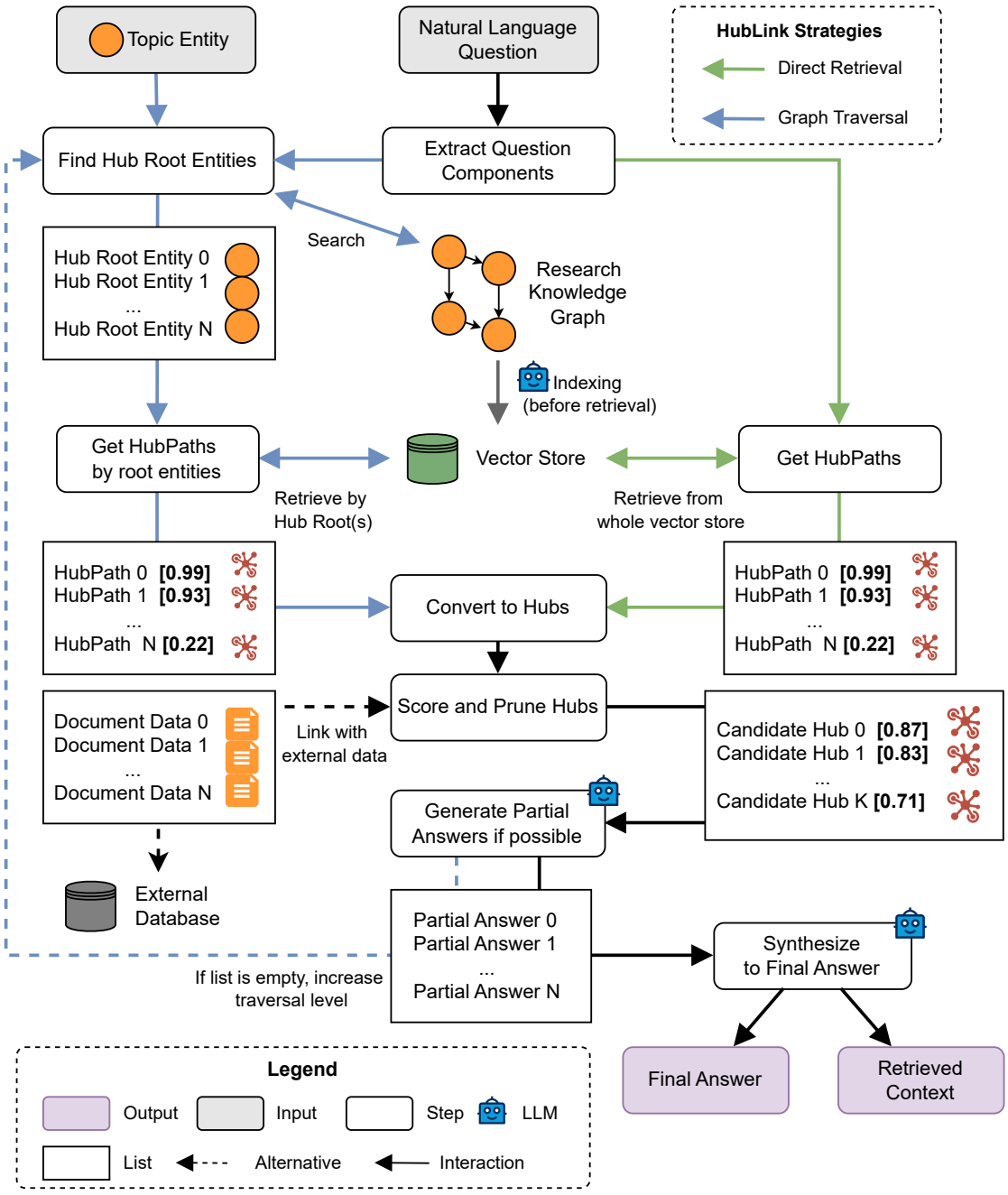


Figure 4.2.: Overview of the retrieval and generation process: Two alternative strategies are depicted. The process of the direct retrieval strategy is shown with green arrows, while the graph traversal retrieval strategy is highlighted in blue.

ANN search is performed on the vector store for each collected root entity to find `HubPaths` that are relevant to the question and contain the root entity in their metadata. This process also involves a deduplication of the paths and the application of a diversity ranker to arrive at the paths that are considered relevant to the question.

Direct Retrieval Strategy: Unlike the graph traversal strategy, the direct strategy does not require a topic entity for retrieval. Furthermore, during query time, this strategy does not impose any additional queries on the graph itself. Therefore, once the components are extracted from the question, the ANN search on the vector store is directly started, which involves querying the whole store instead of querying the hubs directly, as done by the traversal strategy. Before the strategy arrives at the final `HubPaths`, it includes further techniques like clustering by hubs, deduplication of paths, and the application of a diversity ranker.

The result of both strategies up until this point is a list of hubs with n `HubPaths` that the algorithm considers to be the most relevant to answer the question. The next step is to prune the hubs. This is done by aggregating the scores of the paths in each hub by calculating a weighted average. Only the top k hubs with the highest score are kept for the subsequent steps.

In the next step, the *Linking* process of the retriever begins. During this process, additional information is searched in an external database using the identifier of the hub, which further enriches the context of the hub beyond the knowledge collected from the graph. This could be, for example, text passages from the source publications.

By now, all necessary information about each hub has been collected. Next, for each hub, a *partial answer* is generated based on the aggregated information from the hub. This is done by passing the information from the hub together with the original question in a prompt to an LLM. The LLM first checks if it is possible to provide a partial answer to the question based on this information and then provides the partial answer. If the list of partial answers is empty, no answer could be found for the question. At this step, the traversal strategy continues by traversing deeper into the graph. However, for the direct retrieval strategy, the process ends.

If partial answers have been generated, they are consolidated into a final answer. This is done by passing the partial answers to the LLM along with the original question. The LLM then generates the final answer based on this information.

4.2. Formal Definitions

This section lays the formal foundation for the HubLink approach by providing definitions for the key concepts and data structures used throughout the approach. In the following, we first describe the underlying graph model to which the approach is applied. Then, we formally characterize the core elements specific to HubLink: `EntityWithDirection`, `HubRoot`, `HubPath`, `Hub`, and `HubVector`. These definitions are the prerequisites for the detailed algorithmic explanation in the following sections.

4.2.1. Research Knowledge Graph (RKG)

The underlying knowledge base is a Research Knowledge Graph (RKG), which is represented according to the RDF standard [35]. An RDF graph provides a machine-readable semantic framework for describing ontologies, where information is expressed in the form of triples. The graph stores scholarly data where the triples capture not only bibliographic metadata (e.g., authors, dates, publishers) but also detailed relationships between scientific artifacts, for example, contributions of authors or publications. The formal definition of such an RDF graph is provided in Section 2.1 and will not be repeated here.

4.2.2. EntityWithDirection: Directed Graph Node

An entity $e \in E$ represents a node in the RDF graph G, where E is the set of all entities in the graph. Formally, the set of entities is defined as:

$$E = \{e \mid (s, p, o) \in G,\ e \in \{s, o\}\}.$$

We define a path $\mathcal{T}$ as a sequence of n triples, $\mathcal{T} = [t_1, t_2, \ldots, t_n]$ where $n \geq 1$ and $t_i = (s_i, p_i, o_i) \in G$ for all $1 \leq i \leq n$. This sequence represents a directed traversal through the graph G, where adjacent triples are connected via shared entities.

When traversing such a path $\mathcal{T}$, arriving at an entity e via the final triple $t_n = (s_n, p_n, o_n)$ requires understanding the context: specifically, the path taken and whether e served as the subject or object. To capture this context, we introduce the `EntityWithDirection` structure. This structure associates an entity e with the specific path $\mathcal{T}$ that ends at the entity, along with a direction indicator specifying the role of e in the final triple t_n.

Formally, we define an `EntityWithDirection` instance as a tuple $(e, \text{dir}, \mathcal{T})$. This tuple is constructed from a path $\mathcal{T} = [t_1, \ldots, t_n]$, which terminates with the triple $t_n = (s_n, p_n, o_n)$. The components e and dir are determined as follows:

$$(e, \text{dir}) = \begin{cases} (s_n, \rightarrow) & \text{if the path reaches } s_n \text{ as the terminal entity} \\ (o_n, \leftarrow) & \text{if the path reaches } o_n \text{ as the terminal entity} \end{cases}$$

Here, $\rightarrow$ indicates that the entity e is reached as the subject of the final triple, while $\leftarrow$ signifies that the entity e is reached as the object.

We can now define the set $\mathcal{D}_e$ for any given entity $e \in E$, which includes all possible instances of `EntityWithDirection` instances that terminate at this specific entity e:

$$\mathcal{D}_e = \left\{ (e, \text{dir}, \mathcal{T}) \;\middle|\; \mathcal{T} = [t_1, \ldots, t_n], t_n = (s_n, p_n, o_n), (\text{dir} = \rightarrow \wedge e = s_n) \vee (\text{dir} = \leftarrow \wedge e = o_n) \right\}$$

4.2.3. HubRoot: Main Entity of a Hub

A `HubRoot` is an entity of the underlying RKG and serves as the root entity of a hub. These are important for the construction of hubs, as each hub is built by collecting the paths starting from the root entity.

We denote the set of all `HubRoot` entities by $R \subseteq V$, where V is the set of all entities in the graph G. We formally define that the set of all hub roots is given by:

$$R = \{\, v \in V \mid \phi(v) = 1 \,\},$$

where $\phi(v)$ is a binary function to classify an entity as a *HubRoot* if the criteria apply. Formally:

$$\phi(v) = \begin{cases} 1, & \text{if } v \text{ satisfies the hub root criteria,} \\ 0, & \text{otherwise,} \end{cases}$$

Whether $v \in V$ is a member of R is determined by $\phi(v)$. The criteria for defining hub roots can be based on various factors. In the following, we introduce some possibilities:

- **Type-based:** The entity v has a specific type in G. This is useful when the types of entities that are relevant for the QA setting are known prior. For example, in the literature research setting, the objects of interest are publications. In this case, it is straightforward to define publications as hub types.
- **Degree-based:** The degree of the outgoing edges of v exceeds a predefined threshold. This is useful when the information in a graph is highly diverse, making it difficult to define all possible types of hubs.
- **Path-based:** The number of paths at which v acts as the root where the paths reach a certain depth.
- **Semantic-based:** The neighbors of v are checked for semantic similarity. If the similarity exceeds a predefined threshold, v is considered a root of the hub.

Defining the criteria for the hub roots is an ongoing process. We recommend choosing a hybrid approach from the methods suggested above and continually checking the coverage of the graph during indexing. It must be ensured that all entities relevant to the respective QA setting are captured within a hub, as otherwise they cannot be fetched during the retrieval.

4.2.4. HubPath: Path within a Hub

A `HubPath` h_r is built from the paths that start at a `HubRoot` $r \in R$ and lead to an end node. Each `HubPath` consists of a hash value, a textual description, and a list of RDF triples. The hash value is used to uniquely identify the path and is generated by applying a hash function to the list of triples. The textual description is generated by an LLM and constitutes

a natural language representation of h_r. The list of RDF triples represents the path itself and is constructed by traversing the graph starting from the `HubRoot` entity, following the directed nature of the RDF graph, to an end entity.

Formally, let $K \in \mathbb{N}$ be a predetermined maximum path length. The triple path of the `HubPath` $h_{r,i}$ is defined as a sequence

$$\mathcal{T}_{h_{r,i}} = \langle t_1, t_2, \ldots, t_k \rangle, \quad 1 \leq k \leq K,$$

where each element t_j is an RDF triple

$$t_j = \left(v_{j-1}, p_j, v_j\right) \in G,$$

with the convention that the initial entity $v_0 = r$. In this context, v_{j-1} and v_j are entities, and $p_j \in \mathcal{I}$ is the predicate.

The following conditions are imposed on $\mathcal{T}_{h_{r,i}}$:

1. **Acyclic:** The set of entities $\{v_0, v_1, \ldots, v_k\}$ that are included in the path is required to be pairwise distinct.
2. **Single Outbound Degree:** Each intermediate entity v_j (for $1 \leq j < k$) appears exactly once as the object of a triple and exactly once as the subject of a triple.
3. **Termination:** The path terminates when at least one of the following conditions hold:
   a) *End of Graph:* The current entity v_k has no outgoing RDF triple in G.
   b) *New Hub Root:* The current entity v_k is itself a hub root (i.e., $v_k \in R$).
   c) *Maximum Length:* The path has reached the predetermined maximum length, $k = K$.

To preserve semantic continuity, it is important to monitor the length of the triple paths. If the paths are too long, the information from a hub might be too diverse and not semantically connected. If this is the case, reducing the maximum length of the paths can help. An alternative strategy is to adapt the definition of the `HubRoots`, which introduces more hubs and thus can also shorten the overall lengths of the `HubPaths`. However, if the paths are becoming too short, the information might not be detailed enough to answer a question. Therefore, it is important to monitor the lengths of the paths in the graph when building the index.

With the triple path $\mathcal{T}_{h_{r,i}}$, we can now define the hash value of the `HubPath` $h_{r,i}$ as

$$\phi_{r,i} = \text{hash}(\mathcal{T}_{h_{r,i}}) \in \mathcal{B},$$

where $\mathcal{B}$ is the set of all possible hash values. The hash value is generated by applying a hash function to the list of triples in the path. This will produce a unique value for each unique path, ensuring that no two paths have the same hash value, provided that the hash

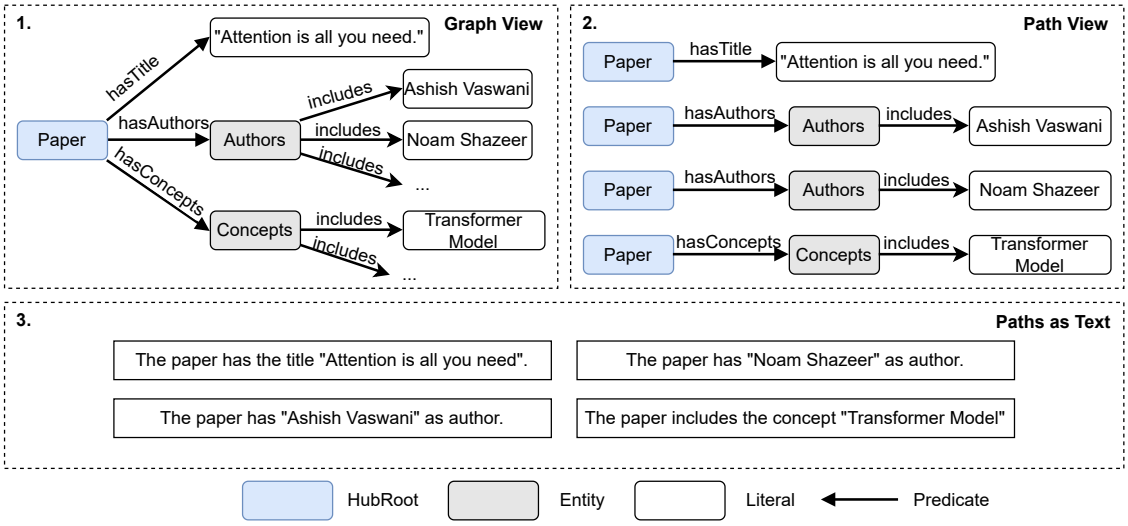


Figure 4.3.: Conversion of paths consisting of triples to textual descriptions using an LLM.

function is free from collisions. We then obtain the textual description of the `HubPath` $h_{r,i}$ by applying a function:

$$T_{\text{path}} : \mathcal{T}_{h_{r,i}} \to \mathcal{X},$$

This function takes a triple path as input and outputs a textual description in a space $\mathcal{X}$ of natural language strings. In Figure 4.3, this process is illustrated. It shows the initial graph view and the decomposition of the hub subgraph into individual triple paths. These are then converted into a textual representation using the function T_{path} which is realized by an LLM that processes the sequence of RDF triples $\mathcal{T}_{h_{r,i}}$ and generates a natural language description.

The final `HubPath` $h_{r,i}$ is then defined as a tuple

$$h_{r,i} = \left(\phi_{r,i}, T_{\text{path}}(\mathcal{T}_{h_{r,i}}), \mathcal{T}_{h_{r,i}}\right),$$

where $\phi_{r,i}$ is the hash value, $T_{\text{path}}(\mathcal{T}_{h_{r,i}})$ is the textual description, and $\mathcal{T}_{h_{r,i}}$ is the list of triples. The `HubPath` $h_{r,i}$ is thus a representation of the path information inside a hub. This leads us to define the set of all `HubPaths` starting from the root of the hub r. Formally, let $r \in R$ be the root of the hub, where R is the set of `HubRoots` as defined in the previous section. We define the set of all HubPaths starting at r as

$$\mathcal{H}(r) = \{\, h_r \mid h_r \text{ is a HubPath starting at } r \,\}.$$

4.2.5. Hub: Aggregation of Semantically Related Information

A `Hub` is a special construct in the HubLink retriever that is used to aggregate semantically related information about a specific knowledge entity. It always consists of a `HubRoot` entity and a set of `HubPaths`. Formally, a hub is defined as

$$Hub_r = \left(r, \mathcal{H}(r)\right),$$

where $r \in R$ is a hub root and $\mathcal{H}(r)$ is the collection of hub paths starting from r.

4.2.6. HubVector: Encoding of Hub Information

To be able to efficiently retrieve relevant information from a hub, the `HubPaths` of the hub are encoded in a vector space using `HubVectors`. A `HubVector` is a low-dimensional vector representation constructed from the information of a `HubPath` and is always stored together with additional metadata that includes the hub root identifier $r \in R$ and the textual description $T_{\text{path}}(\mathcal{T}_{h_{r,i}})$ of the `HubPath`. The goal is to project the semantic content of each `HubPath` into a vector space using a pre-trained embedding model, where a nearest-neighbor search such as ANN can be performed. What is important to consider here is that the path itself may contain too much information, which can lead to a phenomenon where the relevant information is obscured by the complexity of the path. To reduce this risk, the path is decomposed into different parts, increasing the likelihood of a nearest-neighbor search to find relevant information. More information on this design decision is provided in Section 4.9.

By decomposing the `HubPath`, we define four vector representations. Let $h_{r,i}$ denote the i-th `HubPath` of the hub with root r. We define a general embedding function

$$f : \mathcal{X} \to \mathbb{R}^d,$$

which maps any textual input into a d-dimensional vector. We distinguish four types of `HubVectors` that can be generated from a hub path $h_{r,i}$:

- **Path Vector:** This vector represents the overall semantic content of the `HubPath`. It is obtained by embedding the full natural language description $T_{\text{path}}(\mathcal{T}_{h_{r,i}})$ of the path $\mathcal{T}_{h_{r,i}}$ generated by the LLM:

$$\mathbf{v}_{\text{path}}(h_{r,i}) = f\Big(T_{\text{path}}(\mathcal{T}_{h_{r,i}})\Big) \in \mathbb{R}^d$$

- **Triple Vector:** For this vector type, the triple path $\mathcal{T}_{h_{r,i}}$ is broken down into its individual triples. Each triple $t_k \in \mathcal{T}_{h_{r,i}}$ is converted into a vector representation:

$$\mathcal{V}_{\text{triples}} = \{f(t_k) \mid t_k \in \mathcal{T}_{h_{r,i}}\}, \quad f(t_k) \in \mathbb{R}^d$$

- **Entity Vector:** For this vector type, the triple path $\mathcal{T}_{h_{r,i}}$ is decomposed into all entities contained in the path. For each triple $t_k = (s, p, o) \in \mathcal{T}_{h_{r,i}}$, the entities s and o are extracted. Each entity is then embedded using f:

$$\mathcal{V}_{\text{entities}} = \{f(e) \mid t_k = (s, p, o) \in \mathcal{T}_{h_{r,i}},\ e \in \{s, o\}\}, \quad f(e) \in \mathbb{R}^d$$

- **Predicate Vector:** For this vector type, the hub path $h_{r,i}$ is decomposed into all predicates contained in the path. For each triple $t_k = (s, p, o) \in \mathcal{T}_{h_{r,i}}$, the predicate p is extracted and embedded using f. The embedding for each predicate is then computed as:

$$\mathcal{V}_{\text{predicates}} = \{f(p) \mid t_k = (s, p, o) \in \mathcal{T}_{h_{r,i}}\}, \quad f(p) \in \mathbb{R}^d$$

For each hub path $h_{r,i}$, multiple `HubVectors` are thus stored in the vector store. Each vector is stored as a tuple containing the root of the hub r, the natural language description $T_{\text{path}}(\mathcal{T}_{h_{r,i}})$, and the corresponding vector representation. Formally, we define the set of all `HubVectors` for $h_{r,i}$ as:

$$\begin{aligned}\mathcal{V}(h_{r,i}) = &\left\{\left(r,\ T_{\text{path}}(\mathcal{T}_{h_{r,i}}),\ \mathbf{v}_{\text{path}}(h_{r,i})\right)\right\} \\ &\cup \left\{\left(r,\ T_{\text{path}}(\mathcal{T}_{h_{r,i}}),\ v\right) \mid v \in \mathcal{V}_{\text{triples}}\right\} \\ &\cup \left\{\left(r,\ T_{\text{path}}(\mathcal{T}_{h_{r,i}}),\ v\right) \mid v \in \mathcal{V}_{\text{entities}}\right\} \\ &\cup \left\{\left(r,\ T_{\text{path}}(\mathcal{T}_{h_{r,i}}),\ v\right) \mid v \in \mathcal{V}_{\text{predicates}}\right\}\end{aligned}$$

Note that each vector is stored alongside the textual path description $T_{\text{path}}(\mathcal{T}_{h_{r,i}})$ because decomposing the path into individual triples, entities, or predicates results in a loss of the semantic relationships between nodes. This can lead to reduced retrieval performance when a query matches a triple, entity, or predicate, as the isolated vector may not capture enough context to represent the full meaning. To address this, the textual description of the path $T_{\text{path}}(\mathcal{T}_{h_{r,i}})$ is stored as metadata within each vector tuple. While the vector is used for similarity-based retrieval, the associated textual description is used during answer generation.

Finally, given a hub $Hub_r = (r, \mathcal{H}(r))$, the overall set of `HubVectors` for the hub is defined as:

$$\mathcal{V}(Hub_r) = \bigcup_{h \in \mathcal{H}(r)} \mathcal{V}(h).$$

This collection of vectors, together with their associated metadata, forms the basis for the retrieval process in HubLink.

4.3. Data Models

The data models used in the HubLink retriever are shown in Figure 4.4. Each class corresponds to a formal element defined in Section 4.2, with the mapping expressed using the notation Class ← Formal Definition. In the following, we provide a description of each data model:

Class: `Entity` ← $e \in E$ ▷ Represents a node e in the graph
ID: `String`
Value: `String`
Class: `Triple` ← $t = (s, p, o) \in G$ ▷ Represents a triple in the graph
Subject: `Entity`
Predicate: `String`
Object: `Entity`

Class: `EntityWithDirection` ← $(e, \text{dir}, \pi) \in \mathcal{D}_e$
Entity: `Entity` ▷ An entity in the graph
Direction: `Enum` ▷ Indicates if the entity is subject or object in the current path
PathToEntity: `List` of `Triple` ▷ The directed triple path leading to the entity

Class: `Hub` ← $Hub_r = (r, \mathcal{H}(r))$
Root: `EntityWithDirection` ▷ The root node r of the hub
Paths: `List` of `HubPath` ▷ All paths in the hub rooted at r
Score: `Float` ▷ Combined score for all paths

Class: `HubPath` ← $h_{r,i} \in \mathcal{H}(r)$
PathText: `String` ▷ Textual description of the path
PathHash: `String` ▷ Unique hash of the path
Path: `List` of `Triple` ▷ Sequence of triples representing the path
EmbeddedText: `String` ▷ The text used to create the embedding
Score: `Float` ▷ Relevance score toward the query

Class: `ProcessedQuestion`
Question: `String` ▷ The original question that is asked
Components: `List` of `String` ▷ Components of the original question
Embeddings: `List` of `List` of `Float` ▷ Embeddings of components and question

Class: `LinkData`
HubIdentifier: `String` ▷ Identifier associated with the data
Text: `String` ▷ Additional data to enrich the hub

Figure 4.4.: The data models used by HubLink mapped to their formal definitions (Data model ← Formal definition).

Entity The entity data model represents a node in the graph. Each entity is associated with an identifier that uniquely identifies the entity in the graph, is associated with a value, and may also have a type. Depending on the graph, this type can be either stored directly with the entity or described using an edge to the entity. For HubLink, it is not necessary that an entity has a type, but it is advantageous for the classification of hubs.

Triple The triple data model conforms to the RDF definition. As such, it contains a subject and an object, which are both `Entity` types. The predicate describes their relation and is a `String`.

EntityWithDirection This data model stores a direction and a path to an entity. Consequently, it acts as a wrapper for an `Entity` in the HubLink algorithm.

Hub The data model for a hub includes a root, which is an `EntityWithDirection` object. This root acts as a unique identifier for the hub itself and is used for building the `HubPath` objects. In addition, each hub has a score that is calculated during the retrieval to assess the relevance of the hub to the question that is asked.

HubPath The data model for the `HubPath` stores the unique hash value that acts as an identifier for the `HubPath`. The textual description and the associated triple path are also stored. In addition, the model stores the text by which the path has been retrieved from the vector store. This is required as each `HubPath` is stored at different levels (path, triple, entity, or predicate level). Finally, the score that assesses the relevance of the `HubPath` to the question asked is saved in the data model.

ProcessedQuestion This data model is used to store the question that is asked to the retriever, the components that have been extracted from the question, and the precomputed embeddings for both the question and the components.

LinkData This data model is created during the linking process. It stores the identifier of the hub to which it is associated and a text. The text is additional information collected from an external database that is relevant to enrich the retrieval data for comprehensive answer generation.

4.4. Finding Hub Root Entities

The function FINDHUBROOTS, shown in Algorithm 1, is responsible for identifying the `HubRoot` entities of hubs starting from a list of input entities $\vec{E}$, which serve as the initial traversal points within the graph G. The function follows a breadth-first search strategy, meaning that all neighboring entities are explored before descending to a deeper level. To begin the search, a queue is initialized with $\vec{E}$. Then, a loop is performed that terminates once the queue is empty. In each iteration, one entity e is dequeued and processed. If e has already been visited (as tracked in the `visited` set), it is skipped. Otherwise, it is marked as visited, and the function `isHubRoot` is called to determine whether the entity can be classified as a `HubRoot`. If e is identified as a `HubRoot` and has not already been added, it is added to the set *hub_roots* to be returned once the search has finished. Next, a determination for the continuation of the search is performed. If e is a `HubRoot` and its traversal direction is `Right`, the search along that path is terminated, and the algorithm proceeds to the next entity. This is because the `HubPaths` of a hub are built following this

Algorithm 1 Pseudocode for Finding Hub Root Entities

Persistent State:

visited: `Set` of `EntityWithDirection` ▷ List of visited entities

G: `KnowledgeGraph` ▷ The graph to search in

Input:

$\vec{E}$: `List` of `EntityWithDirection` ▷ Entities to start the search from

`UniformHopCount`: `Boolean` ▷ Require uniform hop count when reaching the hubs

Output:

A tuple (`hub_entities`, `next_entities`) containing the entities that are the root of a Hub and the entities that are used to initialize the search on the next level.

```
1:  function findHubRoots(Ē, UniformHopCount)
2:      Initialize queue ← Queue(Ē)
3:      hub_roots ← ∅                          ▷ Collection of Hub root entities found
4:      next_entities ← ∅                      ▷ Candidates returned for the next depth
5:      while queue ≠ ∅ do
6:          e ← dequeue(queue)                 ▷ Current entity to process
7:          if e ∈ visited then Continue
8:          visited ← visited ∪ {e}
9:          e_isHubRoot ← isHubRoot(e)         ▷ Check if entity is a hub
10:         if e_isHubRoot and e ∉ hub_roots then
11:             hub_roots ← hub_roots ∪ {e}
12:         if e_isHubRoot and ¬e.Left then
13:             Continue                       ▷ Stop traversing the current path
14:         entity_triples ← {All adjacent triples of e considering its direction}
15:         if not UniformHopCount then
16:             enqueue all entity_triples into queue
17:         else
18:             next_entities ← next_entities ∪ entity_triples
19:     return (hub_roots, next_entities)
```

direction, which means that those paths will be collected later during the building of the hubs. For all other cases, the function collects the triples in which e participates. These are either queued for further traversal or stored in the set *next_entities*, depending on the parameter `UniformHopCount`. The loop continues until the queue is empty. Once the loop ends, the function returns two sets: *hub_roots*, which contains the discovered hub entities, and *next_entities*, which are passed to the next level of traversal in the indexing process.

Parameter: `UniformHopCount`

This parameter determines whether the traversal should continue beyond the first iteration and is only relevant for the *graph traversal* retrieval strategy. If `UniformHopCount` = `False`, all `HubRoots` directly reachable from the current entities in $\vec{E}$ are considered. Otherwise, the traversal continues recursively until a `HubRoot` is found. This is useful when `HubRoots` lie at varying depths in the graph and a uniform number of hops to reach the entity cannot be assumed. Therefore, it should be chosen whether to set `UniformHopCount` or not based on the structure of the graph.

Persistent State: `visited`

The `visited` set keeps track of entities that have already been processed, taking the direction into account. This ensures that nodes are not revisited multiple times and that cycles in the graph do not cause infinite loops. Notably, this set is persistent for subsequent calls to ensure that cycles are handled for the same retrieval or indexing. Consequently, this persistent state must be cleared once the retrieval or indexing is complete, which can be accomplished by implementing the function in a class and initializing the class for each new retrieval or indexing.

Function: isHubRoot

Whether an entity is classified as a hub is determined by the isHubRoot function. There are several ways to do this classification, as described in Section 4.2. The easiest way is to define a list of types that should be classified as a hub and check if the given entity is of that type.

4.5. Indexing

The pseudocode for the indexing process is shown in Algorithm 2. As input, the function requires a list of starting entities $\vec{E}$ and the parameter `MaxIndexingDepth`, which controls how deep the algorithm traverses G, and thus how large $\hat{G}$ becomes, which is the indexed subgraph. This design choice is motivated by the potential size of the graph G, which can contain millions of nodes. Using the starting entities $\vec{E}$ and the maximum indexing depth `MaxIndexingDepth`, it allows to effectively choose between indexing the entire graph or only a subgraph $\hat{G}$.

The indexing algorithm consists of a loop that terminates when one of the following conditions is met:

1. `MaxIndexingDepth` is reached,
2. the list of entities to process becomes empty, or
3. no further nodes are found for traversal.

Algorithm 2 Pseudocode for Indexing of Hubs

Persistent State:

G: `KnowledgeGraph` ▷ The graph to search in

Input:

$\vec{E}$: `List` of `Entity` ▷ Entities from which the indexing is started

`MaxIndexingDepth`: `Integer` ▷ Maximum allowed indexing depth

1: **procedure** HUBINDEXING($\vec{E}$, `MaxIndexingDepth`)
2: $entities \leftarrow \{\texttt{EntityWithDirection}(T, \textbf{true}, \emptyset) \mid T \in \vec{E}\}$
3: $entities \leftarrow entities \cup \{\texttt{EntityWithDirection}(T, \textbf{false}, \emptyset) \mid T \in \vec{E}\}$
4: **if** $entities = \emptyset$ **then**
5: **return**
6: $depth \leftarrow 0$
7: **while** $depth \leq$ `MaxIndexingDepth` **do**
8: $hub\_roots, next\_entities \leftarrow$ FINDHUBROOTS($entities$)
9: $entities \leftarrow next\_entities$
10: $hub\_end\_entities \leftarrow$ BUILDHUBS(hub_roots)
11: $entities \leftarrow entities \cup hub\_end\_entities$
12: **if** $entities = \emptyset$ **then**
13: **break** ▷ Stop if no more entities to process
14: $depth \leftarrow depth + 1$

In each iteration, the algorithm calls the function FINDHUBROOTS to identify `HubRoot` objects at the current depth, as well as a new set of entities to explore in the next iteration. Once found, the hub roots are passed to the BUILDHUBS function, which builds the `Hub` objects. Then, to prepare the next iteration, the entities located at the ends of these hubs are added to the list of entities used to continue the indexing in the next iteration of the loop.

Parameter: `MaxIndexingDepth`

This parameter defines the maximum traversal depth at which `HubRoot` objects are located. It is important to note that this depth does not correspond to the entities of the graph themselves, but only those classified as `HubRoots`. Since building a hub involves traversing internal paths within the hub itself, the total depth of entities indexed may exceed `MaxIndexingDepth`.

4.5.1. Building Hubs

A hub $Hub_r\big(r, \mathcal{H}(r)\big)$ consists of the root r and the `HubPaths` $\mathcal{H}(r)$. Consequently, the building process is two-fold. First, the triple paths of a hub have to be extracted from the graph, and then the `HubPath` objects have to be created. Then, the paths have to be stored in the vector store.

Algorithm 3 Pseudocode for Building Hubs

Persistent State:
$\hat{V}$: `VectorStore` ▷ Persistent storage for hub-related data
Input:
$\vec{R}$: `List` of `Entity` ▷ Root entities of the hubs
Output:
A list of entities that are used for the next indexing depth.

```
1:  function buildHubs(R⃗)
2:      next_entities ← ∅                          ▷ Entities for deeper indexing
3:      for all r ∈ R⃗ do
4:          (p⃗, e⃗) ← findPathsInHub(r)
5:          next_entities ← next_entities ∪ e⃗
6:          needs_rebuild ← hubIndexNeedsToBeUpdated(r, p⃗)
7:          if needs_rebuild then
8:              hub_paths ← buildHubPaths(p⃗)
9:              storeHubPaths(hub_paths, r)
10:     return next_entities
```

The function for building hubs is detailed in Algorithm 3. This function takes as input a list of root entities $\vec{R}$, each of which is assumed to be confirmed `HubRoot` objects. The algorithm then iterates over every entity $r \in \vec{R}$ to construct the corresponding hub. This is realized in a loop that processes each root entity r. It calls the function findPathsInHub that identifies and returns all triple paths originating from r that are relevant for building the `HubPath` objects. The function further returns a set of additional entities that may serve as roots for nested hubs, which are collected in the set *next_entities* for further indexing. Next, the function hubIndexNeedsToBeUpdated is invoked to determine whether the current hub must be created or updated. If this is the case, buildHubPaths is called to generate a new set of `HubPath` objects based on previously identified paths. After the paths have been collected, the storeHubPaths function is called, which adds them to the vector store.

Persistent Storage: Vector Store $\hat{V}$

The vector store $\hat{V}$ serves as persistent storage for all indexed `HubPath` objects. As mentioned in Section 4.2, it stores four types of vectors on the path, triple, entity, and predicate level. During indexing, the vector store is filled with the vectors, and during retrieval, similarity matches to the vectors are performed to find those `HubPath` objects that are most relevant to the query.

Algorithm 4 Pseudocode for Determining Index Update

Persistent State:
$\hat{V}$: `VectorStore` ▷ A vector store to store hub data

Input:
r: `EntityWithDirection` ▷ The root entity of the hub
$\vec{p}$: `List` of `List` of `Triple` ▷ The collected paths for the hub

Output:
`Boolean` indicating whether the hub data in the vector store needs to be updated

```
1:  function hubIndexNeedsToBeUpdated(r, p⃗)
2:      δ ← computeHashesForPaths(p⃗)
3:      γ ← getStoredPathHashes(r)
4:      if δ ≠ γ then
5:          return true
6:      else
7:          return false

8:  function getStoredPathHashes(r)
9:      v ← V̂.getAllPathVectors(r)
10:     γ ← ∅
11:     for all vector ∈ v do
12:         h ← vector.metadata.pathHash
13:         γ ← γ ∪ {h}
14:     return γ

15: function computeHashesForPaths(p⃗)
16:     δ ← ∅
17:     for all p ∈ p⃗ do
18:         δ ← δ ∪ {hash(p)}
19:     return δ
```

4.5.2. Checking for Graph Updates

Algorithm 4 provides pseudocode to verify whether the index of a hub stored in the vector store $\hat{V}$ is still consistent with the current state of the graph. It plays a key role in ensuring that the index remains consistent with the underlying graph without unnecessarily rebuilding hubs that are already up-to-date.

The check is performed by the function hubIndexNeedsToBeUpdated, which takes as input the root entity r of a hub and a list of current triple paths $\vec{p}$. The function operates under the assumption that the paths in $\vec{p}$ represent the latest structure of the graph. To verify consistency, it first computes a hash value for each triple path and stores the resulting set in δ via the function computeHashesForPaths. These hashes serve as a signature of the current hub structure. Next, the function getStoredPathHashes retrieves the stored

path hashes from the vector store $\hat{V}$ for the given root entity r. These are collected in the set γ, which represents the last known indexed state of the hub. Finally, the sets δ and γ are compared. If they differ, it means that the indexed hub is outdated and needs to be rebuilt. In this case, the function returns `true`. Otherwise, it returns `false`.

Using hash values to compare hub path structures offers an efficient and scalable way to detect changes without needing to perform expensive graph diffing or structural comparisons. Each triple path is mapped to a compact, fixed-size hash, which acts as a unique signature of its content. This allows the algorithm to perform fast set-based comparisons between the current state of the graph and the previously stored hub data. As long as the hash function is consistent and resistant to collision, this approach reliably identifies when any part of the path structure of a hub has changed, thus triggering a rebuild only when necessary.

4.5.3. Finding the Triple Paths of a Hub

To find all relevant triple paths associated with a hub, the function FINDPATHSINHUB, shown in Algorithm 5, is used. This function is invoked with the root entity R of a hub, the knowledge graph G, and the parameter `MaximumPathLength`. It returns all valid triple paths starting at R, together with a list of candidate entities that mark the boundary of the hub for further indexing.

At the start of the function, three sets are initialized: `visited`, `candidates`, and `paths`. The `visited` set tracks entities already explored to prevent cycles. The `candidates` set stores entities that are themselves hubs and may serve as roots for nested hubs in later indexing stages and the `paths` set contains all complete triple paths discovered within the hub.

A queue is initialized with a tuple containing the root entity R and an empty path. The function performs a breadth-first traversal using this queue. In each iteration, an entity e and its associated path $\vec{p}$ are dequeued. If e has already been visited and is not the root, it is skipped. If the current path length has reached the maximum allowed length, the path is stored, and traversal for that branch ends. Otherwise, the function retrieves all outbound triples of e. If there are no outbound triples, the node is considered a leaf, and the current path is saved. However, if there are outbound triples, each triple is examined. This is done by extracting the entity object e' from the triple. The reason we are only looking at the objects of the RDF triples is that, by convention, the paths of a hub are only examined in the natural direction of the graph. The algorithm then continues by checking whether e' is the `HubRoot` of the hub R. In this case, it is skipped to avoid cycles. If e' is identified as a `HubRoot` for another hub, it is added to the `candidates` set, and the current path is stored without traversing further in that direction. Otherwise, the triple is appended to the current path and the new state $(e', \vec{p}')$ is enqueued for further exploration. This process continues until the queue is empty. At that point, all discovered paths and candidate hub boundaries have been collected and are returned by the function.

Algorithm 5 Pseudocode for Finding the Triple Paths of a Hub

Persistent State:

G: `KnowledgeGraph` ▷ The graph to search in

Input:

R: `Entity` ▷ Root entity of the hub

`MaximumPathLength`: `Integer` ▷ Maximum allowed path length

Output:

A tuple $(paths, candidates)$, where $paths$ are the collected triple paths, and $candidates$ are entities at hub boundaries

1: **function** FINDPATHSINHUB(R, `MaximumPathLength`)
2: $visited \leftarrow \emptyset$ ▷ Set of entities visited
3: $candidates \leftarrow \emptyset$ ▷ Candidates for next level
4: $paths \leftarrow \emptyset$ ▷ Paths of triples from the graph
5: $queue \leftarrow$ a new queue initialized with $(R, \emptyset)$
6: **while** $queue \neq \emptyset$ **do**
7: $(e, \vec{p}) \leftarrow$ DEQUEUE($queue$) ▷ Current entity and path
8: **if** $e \in visited$ and $e \neq R$ **then**
9: **continue** ▷ Avoid revisiting entities
10: $visited \leftarrow visited \cup \{e\}$
11: **if** $|\vec{p}| \geq$ `MaximumPathLength` **then**
12: $paths \leftarrow paths \cup \{\vec{p}\}$
13: **continue**
14: $\vec{t} \leftarrow$ GETOUTBOUNDTRIPLES(e, G) ▷ Retrieve all neighbors of e
15: **if** $\vec{t} = \emptyset$ **then**
16: $paths \leftarrow paths \cup \{\vec{p}\}$ ▷ Leaf entity of graph reached
17: **continue**
18: **for all** $triple \in \vec{t}$ **do**
19: $e' \leftarrow triple.Object$
20: **if** $e' = R$ **then**
21: **continue** ▷ Skip cycles back to root
22: **if** ISHUBROOT(e') **then**
23: $candidates \leftarrow candidates \cup \{e'\}$
24: $paths \leftarrow paths \cup \{\vec{p}\}$
25: **else**
26: $\vec{p}' \leftarrow \vec{p}$ ++ $\{triple\}$ ▷ Append triple to path
27: enqueue $(e', \vec{p}')$ into $queue$ ▷ Add neighbor to queue
28: **return** $(paths, candidates)$

Parameter: `MaximumPathLength`

The parameter `MaximumPathLength` limits how deeply the algorithm explores the graph from the `HubRoot` R. It defines the maximum allowed length of any `HubPath` and helps control the scope and relevance of each path. A path that is too long may lose semantic connection to the core context of the hub. Therefore, choosing an appropriate value for `MaximumPathLength` depends on the structure and density of the graph.

Algorithm 6 Pseudocode for Building HubPaths

Input:
$\vec{T}$: `List` of `List` of `Triple` ▷ The paths to process

Output:
A list of `HubPath` objects.

```
1: function buildHubPaths(T⃗)
2:     hub_paths ← ∅
3:     for all t⃗ ∈ T⃗ do
4:         δ ← hash(t⃗)                          ▷ Calculate unique path identifier
5:         𝒯 ← pathToText(t⃗)
6:         hub_paths ← hub_paths ∪ {HubPath(𝒯, δ, t⃗)}
7:     return hub_paths

8: function pathToText(t⃗)
9:     return Convert t⃗ to text using an LLM
```

4.5.4. Building the HubPaths of a Hub

As defined in Section 4.2, a `HubPath` consists of a hash value, a natural language description, and a path of triples. The function buildHubPaths, shown in Algorithm 6, is responsible for generating the `HubPath` objects associated with a hub. It takes as input a list $\vec{T}$, where each element $\vec{t} \in \vec{T}$ is a path composed of RDF triples originating from the root. For each path $\vec{t}$, a unique identifier δ is computed by hashing the content of the path. This hash serves as an identifier for identifying the `HubPath` object. The path is then converted into a textual representation $\mathcal{T}$ using the function pathToText, which uses an LLM to generate a natural language description of the path. The resulting `HubPath` object is then instantiated using the hash δ, the textual description $\mathcal{T}$, and the original triple path $\vec{t}$. All such `HubPath` objects are collected and returned as the output of the function.

Algorithm 7 Pseudocode for Storing HubPaths in the Vector Store

Persistent State:
$\hat{V}$: `VectorStore` ▷ A vector store to store hub data

Input:
$\vec{H}$: List of `HubPath` ▷ Hubpath objects to store in the vector store
r: `HubRoot` ▷ Root entity of the hub the paths are associated with

procedure storeHubPaths($\vec{H}, r$)
  $\hat{V}$.deleteHubPaths(r) ▷ Delete existing paths of the root entity r
  for all $h \in \vec{H}$ **do**
    $metadata \leftarrow (r, h.PathHash, h.PathText, h.Path)$ ▷ Metadata for the vectors
    storePath($h.PathText, metadata$) ▷ Store the path vector
    storeTriples($h.Path, metadata$) ▷ Store the triples in the vector store
    storeEntities($h.Path, metadata$) ▷ Store the entities in the vector store
    storePredicates($h.Path, metadata$) ▷ Store the predicates in the vector store

procedure storePath($\mathcal{T}$, $metadata$)
  addToVectorStore($\mathcal{T}, metadata$)

procedure storeTriples($\vec{t}, metadata$)
  for all $triple \in \vec{t}$ **do**
    addToVectorStore($triple.subject, metadata$)

procedure storeEntities($\vec{t}, metadata$)
  $entities \leftarrow \emptyset$
  for all $triple \in \vec{t}$ **do**
    $entities \leftarrow entities \cup \{triple.subject\}$
    $entities \leftarrow entities \cup \{triple.object\}$
  for all $e \in entities$ **do**
    addToVectorStore($e, metadata$)

procedure storePredicates($\vec{t}, metadata$)
  $predicates \leftarrow \emptyset$
  for all $t \in \vec{t}$ **do**
    $predicates \leftarrow predicates \cup \{t.predicate\}$
  for all $p \in predicates$ **do**
    addToVectorStore($p, metadata$)

procedure addToVectorStore($itemToEmbed, metadata$)
  $\vec{\mu} \leftarrow$ getEmbedding($itemToEmbed$) ▷ Embed the text
  $\delta \leftarrow$ hash($object$) ▷ Hash the text to get a unique key
  $etadata \leftarrow metadata$ ++ $itemToEmbed$ ▷ Add embedded text to metadata
  $\hat{v}$.store($\delta, \vec{\mu}, metadata$) ▷ Add the vector by key with the metadata

4.5.5. Storing the HubPath Objects

Algorithm 7 presents the pseudocode of the storeHubPaths function, which stores `HubPath` objects in the vector store $\hat{V}$. The function takes as input a list of `HubPath` objects and a `HubRoot` entity r. It first deletes any previously stored `HubPath` objects associated with r from the vector store $\hat{V}$ to ensure that outdated representations do not persist across indexing runs. Then, for each `HubPath` h, the function prepares metadata, including the root entity, a path hash, a natural language description, and the triple path itself. This metadata is then used in a series of helper functions, each responsible for storing a specific vector type, as defined in Section 4.2.

There are various helper functions involved for storing the `HubPath` at different vector levels. The helper function is addToVectorStore which is used by all the subsequent storing methods. This function receives as input any text to then embed it into a vector, creates a hash value that acts as a unique key, and stores the embedding with the metadata at the location in the vector store associated with the key. The following helper functions are used for storing the `HubPath` at different vector levels. First, storePath stores the textual description of the `HubPath`. Then, the function storeTriples extracts all triples from the path and stores each one separately. Next, storeEntities collects all subjects and objects from the triples and storePredicates extracts all predicates.

Path Hash: Unique Vector Key

Hashing the path to a unique key allows efficient storage and updating of vector entries in the vector store. Because the hash depends on the content of the path, any structural change will result in a different hash. This ensures that updates to paths can be recognized during an update of the index without having to recalculate the entire index.

4.6. Vector Store Retrieval

During the indexing process, the `HubPath` objects of a `Hub` have been stored in a vector store. This section details the algorithms for retrieving these `HubPath` objects. Specifically, we first outline the procedures for their retrieval from the vector store before explaining the diversity-based ranking algorithm applied to diversify the retrieval.

4.6.1. Global Retrieval of HubPaths

The function globalSimilaritySearch, shown in Algorithm 8, is responsible for retrieving the most relevant hubs from the vector store $\hat{V}$ based on a given question. It takes as input a `ProcessedQuestion` object $\hat{Q}$, a list of `Hub` objects to exclude $\vec{H}$, and the number of top paths to keep per hub, specified by the parameter `TopPathsToKeep`.

Algorithm 8 Pseudocode for Global HubPath Retrieval

Persistent State:
$\hat{V}$: `VectorStore` ▷ Stores all vector representations for hub paths
Input:
$\hat{Q}$: `ProcessedQuestion` ▷ The processed question
$\vec{H}$: `List` of `Hub` ▷ Hubs excluded from the search
`TopPathsToKeep`: `Integer` ▷ Number of paths considered for each Hub
Output:
A list of `Hub` objects most relevant to the question

function GLOBALSIMILARITYSEARCH($\hat{Q}, \vec{H}$, `TopPathsToKeep`)
  $excluded \leftarrow \{h.Root \mid h \in \vec{H}\}$ ▷ Hubs excluded from the search
  $filter \leftarrow$ `hub_entity` $\notin excluded$ ▷ Filter for the vector store query
  $n\_results \leftarrow$ `TopPathsToKeep` $\cdot\, 2$ ▷ Number of hub paths to return
  $\vec{\mu} \leftarrow \emptyset$ ▷ Set of unique path hashes
  while $|\vec{\mu}| < n\_results$ **do**
    $\vec{\phi} \leftarrow \hat{V}$.QUERY($\hat{Q}.Embeddings, n\_results, filter$)
    if $\vec{\phi} = \emptyset$ **then**
      break
    $paths \leftarrow$ empty map from `Entity` to `List` of `HubPath`
    for all $\phi \in \vec{\phi}$ **do**
      $hub\_path \leftarrow$ CONVERTTOHUBPATH(ϕ) ▷ Convert result to HubPath object
      if $hub\_path.PathHash \in \vec{\mu}$ **then**
        continue ▷ Skip duplicate paths
      $hub\_entity \leftarrow \phi.metadata.RootEntity$
      $paths[hub\_entity] \leftarrow paths[hub\_entity] \cup \{hub\_path\}$
      $\vec{\mu} \leftarrow \vec{\mu} \cup \{hub\_path.PathHash\}$
  $final\_hubs \leftarrow \emptyset$
  for all $(root, paths) \in paths$ **do**
    $ranked \leftarrow$ APPLYSUBJECTDIVERSITYRANKER($paths$)
    $pruned \leftarrow$ PRUNEPATHS($ranked$, `TopPathsToKeep`)
    $hub \leftarrow$ `Hub`$(root, pruned)$
    $final\_hubs \leftarrow final\_hubs \cup \{hub\}$
  return $final\_hubs$ ▷ Return the ranked hubs

The function starts by constructing a filter to exclude all root entities of the hubs already contained in $\vec{H}$. Then, it proceeds with an iterative loop. The loop starts by querying the vector store with the embeddings of $\hat{Q}$, asking for the $2 \cdot$`TopPathsToKeep` most similar vectors that conform to the filter. This search is an ANN search that ranks the results according to the similarity of the question vector with the vectors stored in the database. The reason we are multiplying the number of retrieved paths by two is to ensure a sufficient candidate pool for the following diversity-based ranking. If no results are returned for the query, the loop is terminated early. Otherwise, it initializes a mapping called `paths`, which groups retrieved paths by their associated hub entities. Each result ϕ is processed by extracting the root of

the hub from the metadata and converting the vector representation into a `HubPath` object. Subsequent to the conversion process, a verification procedure is initiated to determine whether the specified path has been previously added. This verification is conducted by comparing the hash value of the given path with those stored in a list of unique hash values. In the event that the path has not been previously added, it is appended to the map at the entry of its root. Following this, the loop progresses if not enough paths have been extracted from the vector store. Otherwise, the loop is terminated.

Afterwards, the function initializes an empty list called `final_hubs`, which is later returned. For each hub root entity in `paths`, the associated paths are passed to the function applySubjectDiversityRanker (see Section 4.6.3), which ranks them by semantic relevance while promoting subject diversity at the triple level. The ranked list is then pruned to retain only the top `TopPathsToKeep` paths. Finally, a new `Hub` object is created for each root entity using the ranked and pruned paths, and the resulting hubs are collected in `final_hubs`. Once all candidates are processed, the list of hubs is returned.

4.6.2. Hub-Specific HubPath Retrieval

Algorithm 9 presents the pseudocode for retrieving the top `TopPathsToKeep` hub paths associated with a specific root entity r. The getHubPathsForHub function takes as input the root entity r, a processed question $\hat{Q}$, and the number of paths to retain. The output is a ranked and pruned list of `HubPath` objects that are most relevant to the question.

First, the function initializes two collections: $\vec{\mu}$ and `hub_paths`. The former stores the hash values of the already selected paths and the latter stores the actual `HubPath` objects to return. The main loop continues until the number of unique paths collected in $\vec{\mu}$ reaches the desired count, `TopPathsToKeep`. In each iteration, a filter is constructed to exclude already seen path hashes and to restrict the search to paths associated with the current hub entity r. The vector store $\hat{V}$ is then queried for up to $2 \cdot$ `TopPathsToKeep` results matching this filter. This over-retrieval is intentional as it provides a larger candidate pool for the applySubjectDiversityRanker (see Section 4.6.3), which reranks the results to promote subject diversity among the selected paths. If the query returns no results, the loop terminates. Otherwise, the results are converted into `HubPath` objects and passed to the diversity ranker. The ranked paths are then iterated over. If the hash of a path already exists in $\vec{\mu}$, it is skipped to avoid duplicates. This is necessary because each `HubPath` is stored at multiple representation levels in the vector store (see Section 4.2.6), and this process ensures that only the version of each path with the highest initial similarity score remains, thereby ensuring relevance. If the number of retained paths reaches `TopPathsToKeep`, the loop exits early. Otherwise, the current path is added to `hub_paths`, and its hash is added to $\vec{\mu}$. If too many duplicates were encountered in the current iteration, this could lead to a situation where the number of unique paths has not yet reached the desired number. In this case, the loop continues, using the updated filter to retrieve more candidates. Once the loop terminates, the function returns the final list of `HubPath` objects associated with root r that are most relevant to the question.

Algorithm 9 Pseudocode for HubPath Retrieval

Persistent State:
$\hat{V}$: `VectorStore` ▷ Stores all vector representations for hub paths
Input:
r: `Entity` ▷ The root entity of the hub
$\hat{Q}$: `ProcessedQuestion` ▷ The processed question
`TopPathsToKeep`: `Integer` ▷ Number of paths to keep for this hub
Output:
A list of `HubPath` objects associated with root entity r

function GETHUBPATHSFORHUB($r, \hat{Q}$, `TopPathsToKeep`)
  $\vec{\mu} \leftarrow \emptyset$ ▷ Set of unique path hashes
  $hub\_paths \leftarrow \emptyset$ ▷ List of HubPaths to return
  while $|\vec{\mu}| <$ `TopPathsToKeep` **do**
    $filter \leftarrow \{$`path_hash` $\notin \vec{\mu} \wedge$ `hub_entity` $= r\}$
    $n\_results \leftarrow$ `TopPathsToKeep` $\cdot 2$
    $\vec{r} \leftarrow \hat{V}$.QUERY($\hat{Q}.Embeddings, n\_results, filter$)
    if $\vec{r} = \emptyset$ **then**
      break ▷ No more paths to process
    $parsed\_paths \leftarrow$ CONVERTTOHUBPATHS($\vec{r}$) ▷ Convert results to HubPath objects
    $ranked\_paths \leftarrow$ APPLYSUBJECTDIVERSITYRANKER($parsed\_paths$)
    for all $hub\_path \in ranked\_paths$ **do**
      if $hub\_path.PathHash \in \vec{\mu}$ **then**
        continue ▷ Skip duplicate paths
      if $|hub\_paths| \geq$ `TopPathsToKeep` **then**
        break ▷ Reached the limit of paths to keep
      $hub\_paths \leftarrow hub\_paths \cup \{hub\_path\}$
      $\vec{\mu} \leftarrow \vec{\mu} \cup \{hub\_path.PathHash\}$
  return hub_paths

4.6.3. Diversity-Based Ranking of HubPaths

The pseudocode for the function APPLYSUBJECTDIVERSITYRANKER is shown in Algorithm 10. The goal of this function is to promote diversity among the retrieved `HubPath` objects by penalizing paths that share the same subject entity. This is particularly useful in cases where multiple paths are relevant but semantically redundant due to overlapping subjects. The rationale behind this reranking approach is discussed in more detail in Section 4.9.

The function takes as input a list of `HubPath` objects $\vec{h}$ and a diversity penalty parameter `DiversityPenalty`. It begins by sorting the paths in descending order on the basis of their initial scores. Next, the function initializes a subject frequency map `subject_counts`, which tracks how many times each subject appears across the paths. The algorithm then iterates through the sorted list of `HubPath` objects. For each path h, the subject is extracted from the text that was used to embed the path using the function EXTRACTSUBJECT. However,

Algorithm 10 Pseudocode for Diversity Ranking of HubPaths

Input:
$\vec{h}$: `List` of `HubPath` ▷ HubPaths to rerank
`DiversityPenalty`: `Float` ▷ Penalty applied for repeated subjects
Output:
A reranked list of `HubPath` objects promoting subject diversity

function APPLYSUBJECTDIVERSITYRANKER($\vec{h}$, `DiversityPenalty`)
  $pre\_sorted \leftarrow$ SORTBYSCORE($\vec{h}$) ▷ Initial sort by original score
  $subject\_counts \leftarrow$ empty map from subject to count
  for all $h \in pre\_sorted$ **do**
    $text \leftarrow h.EmbeddedText$
    $subject \leftarrow$ EXTRACTSUBJECT($text$) ▷ Extract the subject from the text
    if $subject \neq$ `None` **then**
      $count \leftarrow$ GETORDEFAULT($subject\_counts, subject, 0$)
      $h.Score \leftarrow h.Score - (\texttt{DiversityPenalty} \cdot count)$
      $subject\_counts[subject] \leftarrow count + 1$
  $reranked \leftarrow$ SORTBYSCORE($\vec{h}$) ▷ Re-sort by adjusted scores
  return $reranked$

this extraction only returns a subject if the text that was embedded actually represents a triple. The reason why the text may not be a triple comes from the different content levels at which the `HubPaths` are embedded, which are path, triple, entity, or predicate, as explained in Section 4.2. If a subject is successfully extracted from the text, a penalty is applied to the score of the path based on how often that subject has previously appeared. Specifically, the score is reduced by the product of `DiversityPenalty` and the current count for that subject. After applying the penalty, the count of the extracted subject is incremented in `subject_counts`, ensuring that repeated appearances of the same subject are penalized more heavily on subsequent occurrences. Finally, the function applies a second sorting on the modified list of paths and returns the result.

4.7. Retrieval

The following section describes the retrieval process of the HubLink retriever. This process is responsible for finding the hubs and their associated paths that can be used to generate partial answers that are then consolidated into a final answer. Two strategies for retrieval are proposed. The *Direct Retrieval Strategy* offers faster retrieval with reduced accuracy for local queries, whereas the *Graph Traversal Strategy* improves local query precision but requires more execution time. Both strategies receive a `ProcessedQuestion` object as input, the explanation of which we are going to start in the following.

Algorithm 11 Pseudocode for Creating a ProcessedQuestion

Input:
Q: `String` ▷ The question being asked
Output:
A `ProcessedQuestion` object containing embeddings and components

```
1: function processQuestion(Q)
2:     embeddings ← ∅
3:     q_embedding ← getEmbedding(Q)                ▷ Embed the question
4:     embeddings ← embeddings ∪ {q_embedding}
5:     components ← extractComponents(Q)
6:     for all c ∈ components do
7:         embeddings ← embeddings ∪ {getEmbedding(c)}
8:     return ProcessedQuestion(Q, components, embeddings)

9: function extractComponents(Q)
10:    return Extract components from Q using an LLM
```

4.7.1. Question Processing

The first step in the retrieval process is to preprocess the question Q. This involves two tasks: extracting the semantic components of the question and computing vector embeddings for both the question and the components of the question using an embedding model. This procedure is illustrated in the pseudocode in Algorithm 11, which defines the function processQuestion.

The function receives the question as input and starts by computing an embedding of the entire question using a pre-trained embedding model. Next, the semantic components are extracted from the question using an LLM. By identifying individual components such as entities, types, time expressions, or contextual constraints, the retriever can perform more granular matching during the retrieval phase. This step is especially important for handling complex queries that include multiple constraints or conditions. For example, a question like "Which papers have been published by SpringerLink in 2020?" could be decomposed into the components: `['Publisher', 'SpringerLink', '2020']`. The extraction is performed using an LLM, which parses the question and returns its relevant parts as a list. Each extracted component is then individually embedded using the same embedding model, and all resulting vectors are stored together with the original question embedding in a list. Finally, a `ProcessedQuestion` object is initialized. This object contains the original question, the list of extracted components, and the corresponding vector embeddings. The function then returns this object.

Algorithm 12 Pseudocode for the direct retrieval strategy

Persistent State:
G: `KnowledgeGraph` ▷ The graph to search in

Input:
Q: `String` ▷ The question being asked
`TopPathsToKeep`: `Integer` ▷ Number of paths considered for each Hub
`HubsToKeep`: `Integer` ▷ Number of candidates to collect

Output:
A tuple (`retrieved_triples`, `final_answer`)

```
1:  function directRetrievalStrategy(Q)
2:      Q̂ ← processQuestion(Q)
3:      candidate_hubs ← ∅                          ▷ List of candidate hubs
4:      while |candidate_hubs| < HubsToKeep do
5:          r⃗ ← globalSimilaritySearch(Q̂, candidate_hubs, TopPathsToKeep)
6:          if r⃗ = ∅ then
7:              break
8:          candidate_hubs ← candidate_hubs ∪ r⃗
9:      candidate_hubs ← fillOrRemovePaths(Q̂, candidate_hubs)
10:     final_hubs ← pruneHubs(candidate_hubs)
11:     partial_answers ← ∅
12:     for all hub ∈ final_hubs do
13:         partial ← getPartialAnswerForHub(hub, Q̂)
14:         partial_answers ← partial_answers ∪ {partial}
15:     if partial_answers ≠ ∅ then
16:         return getFinalAnswer(partial_answers, Q)
17:     else
18:         return ( ∅, None)
```

4.7.2. Direct Retrieval Strategy

In this section, the pseudocode for the *direct retrieval* strategy is presented. The direct retrieval strategy directly performs an ANN search on the index stored in the vector store. Unlike the traversal strategy, it does not make any calls to the graph during the retrieval phase. In the following, we are first going to introduce the pseudocode for the main function of the retrieval strategy in Algorithm 12. Then we present the pseudocode for the helper function that is used to ensure the number of paths for each hub in Algorithm 13.

4.7.2.1. Main Function of the Direct Retrieval Strategy

In Algorithm 12 the pseudocode for the directRetrievalStrategy function is shown. It is the main function to realize the direct retrieval strategy, which performs a semantic search

across the indexed KG to identify relevant hubs and generate answers to natural language questions.

The function takes a question Q as input. It begins by invoking the function processQuestion, which computes the semantic embedding of the question and extracts key features, producing a `ProcessedQuestion` object. Next, a loop is executed to iteratively collect candidate hubs until the number specified by `HubsToKeep` is reached. In each iteration, the function globalSimilaritySearch is called with the processed question and the current set of hubs. This function queries the vector store to retrieve a batch of new hub candidates, excluding any candidate already retrieved, and adds them to the `candidate_hubs` set. The loop terminates early if no new results are found. Once a sufficient set of candidates is collected, the function fillOrRemovePaths is called to ensure that each hub contains no more than `TopPathsToKeep` of `HubPath` objects. These paths represent the most semantically relevant information within each hub and are selected based on similarity and subject diversity. The filtered hubs are then passed to the pruneHubs function, which evaluates and ranks them according to their overall relevance to the input question. The highest scoring hubs limited by `HubsToKeep` are selected for processing. Each selected hub is then processed by the function getPartialAnswerForHub, which generates a partial answer based on the content of the hub and the processed question. If partial answers are produced, they are passed to the getFinalAnswer function, which aggregates them into a single coherent response. If no partial answers are generated, the function returns an empty result.

Parameter: `TopPathsToKeep`

The parameter `TopPathsToKeep` defines the maximum number of `HubPath` objects retrieved and used per hub during the retrieval process. It directly controls how much information is considered to generate the partial answer for each hub. A higher value increases the likelihood of including relevant content, potentially improving the quality of the responses. However, it also expands the context window passed to the LLM, which increases computational cost, latency, and may introduce noise.

Parameter: `HubsToKeep`

The parameter `HubsToKeep` determines how many hubs are selected for comparison and partial answer generation. A higher value increases the number of hubs considered, which can improve coverage and quality, especially for complex questions that require information from multiple sources. However, this also increases the number of queries sent to the LLM, leading to higher retrieval costs and runtime. Consequently, the optimal value for `HubsToKeep` depends on the structure of the knowledge graph and the nature of the question.

4.7.2.2. Ensuring the Desired Number of HubPaths

To guarantee that an equal number of paths are considered for each hub, the fillOrRemovePaths function is responsible. The function is presented in Algorithm 13 and ensures

Algorithm 13 Pseudocode to Ensure Desired Number of HubPaths

Persistent State:

G: `KnowledgeGraph` ▷ The graph to search in

$\hat{V}$: `VectorStore` ▷ A vector store to store HubPaths

Input:

$\vec{H}$: `List` of `Hub` ▷ Hubs to fill or prune

$\hat{Q}$: `ProcessedQuestion` ▷ The processed question

`TopPathsToKeep`: `Integer` ▷ Desired number of paths per hub

Output:

A list of `Hub` objects, each with at most `TopPathsToKeep` paths

```
1:  function fillOrRemovePaths(H⃗, Q̂, TopPathsToKeep)
2:      for all h ∈ H⃗ do
3:          paths ← h.Paths
4:          if |paths| > TopPathsToKeep then
5:              pruned ← prunePaths(paths, TopPathsToKeep)
6:              h.Paths ← pruned
7:              continue
8:          if |paths| = TopPathsToKeep then
9:              continue                          ▷ No need to fill or prune the paths
10:         hub_paths ← V̂.getHubPathsForHub(h.HubRoot.Entity, Q̂, TopPathsToKeep)
11:         h.Paths ← hub_paths
12:     return H⃗
```

that for each hub in the list $\vec{H}$ the number of paths is equal to or less than `TopPathsToKeep` `HubPath` objects, either by pruning excess paths or retrieving additional ones.

The fillOrRemovePaths function iterates over each hub $h \in \vec{H}$ and first checks whether the number of paths associated with the hub already exceeds the specified threshold. If so, it calls prunePaths to reduce the number of paths to `TopPathsToKeep`. Otherwise, if the hub already contains `TopPathsToKeep` paths, no action is taken. However, if the hub has fewer than the desired number of paths, the function retrieves additional `HubPath` objects by calling getHubPathsForHub, passing the root entity of the hub, the processed question $\hat{Q}$, and the `TopPathsToKeep` parameter. This retrieval step queries the vector store $\hat{V}$ and returns a ranked list of relevant paths. The retrieved paths are then assigned to the hub. Once all paths grouped by their associated hub have been processed, each hub has at most `TopPathsToKeep` paths, depending on the total number of available paths for the hub. These modified hubs are then returned by the function.

4.7.3. Graph Traversal Retrieval Strategy

In this section, the pseudocode for the *graph traversal* strategy is presented. In addition to the question that is asked, the strategy receives the identifier of an entity in the graph

Algorithm 14 Pseudocode for the Graph Traversal Strategy

Persistent State:
G: `KnowledgeGraph` ▷ The graph to search in

Input:
Q: `String` ▷ The question being asked
T: `Entity` ▷ The topic entity to start from
`MaxRetrievalLevel`: `Integer` ▷ Maximum depth for graph traversal

Output:
A tuple (`retrieved_triples`, `final_answer`)

function graphTraversalRetrievalStrategy(Q, T)
  $roots \leftarrow \{$`EntityWithDirection`$(T, \texttt{true}, \emptyset)$, `EntityWithDirection`$(T, \texttt{false}, \emptyset)\}$
  $level \leftarrow 1$
  $\hat{Q} \leftarrow$ processQuestion(Q)
  while $level \leq$ `MaxRetrievalLevel` **do**
    if $roots = \emptyset$ **then**
      return ($\emptyset$, `None`) ▷ No more entities to traverse
    ($hub\_candidates$, $next\_entities$) $\leftarrow$ getHubsAtCurrentLevel($\hat{Q}$, $roots$)
    $roots \leftarrow next\_entities$
    if $hub\_candidates = \emptyset$ **then**
      continue ▷ No relevant hubs at this level
    $relevant\_hubs \leftarrow$ pruneHubs($hub\_candidates$)
    $partial\_answers \leftarrow \emptyset$
    for all $hub \in relevant\_hubs$ **do**
      $partial \leftarrow$ getPartialAnswerForHub(hub, $\hat{Q}$)
      $partial\_answers \leftarrow partial\_answers \cup \{partial\}$
    if $partial\_answers \neq \emptyset$ **then**
      return getFinalAnswer($partial\_answers$, Q)
    $level \leftarrow level + 1$
  return ($\emptyset$, `None`) ▷ No answer has been found

to use as an entry point for subsequent traversal. We will refer to this entity as the *topic entity*. In the following, we first introduce the main function of the strategy. Subsequently, a helper function is introduced, which identifies the hubs by traversing the graph.

4.7.3.1. Main Function of the Graph Traversal Retrieval Strategy

In Algorithm 14, the pseudocode for the graph traversal retrieval strategy is presented. The function graphTraversalRetrievalStrategy begins by initializing a set of root entities, denoted as `roots`, containing two directional entries for the topic entity: one for forward traversal and one for backward traversal. This bidirectional setup enables exploration in both directions from the entity T. Furthermore, the traversal level is initialized to one and increases for each subsequent iteration. Next, the input question is processed using the

function processQuestion. This produces a `ProcessedQuestion` object which includes the question itself, the components that have been extracted from the question, and the precomputed embeddings.

Once these values are prepared, the main loop is executed that runs until the maximum retrieval level defined by `MaxRetrievalLevel` is reached. At each level, the algorithm first checks whether there are any entities left to explore. This is done by checking whether the list `roots` is empty, which happens if either no topic entity was provided or if during previous iterations no more entities have been collected that can be further traversed. In the case that the list is empty, the process terminates with no result.

Otherwise, if entities can be traversed, the function getHubsAtCurrentLevel is called with those entities and the processed question. This function has the task of finding a list of hub candidates, along with a new set of entities for the next traversal level. If the function is unsuccessful, meaning that no hub candidates can be reached from the provided entities, the next iteration of the outer loop is started. Otherwise, the hub candidates are passed to another function named pruneHubs. This function selects the most relevant hubs by calculating the overall score of the hub. Those top-ranking hubs are then passed to getPartialAnswerForHub, which generates partial answers using the context of the current hub and the original question. If any partial answers are generated, they are aggregated using getFinalAnswer, which synthesizes them into a coherent response. If no partial answers are produced, the algorithm continues to the next iteration.

Parameter: `MaxRetrievalLevel`

The parameter `MaxRetrievalLevel` defines the maximum number of levels (or hops) the traversal strategy will explore in the graph. Increasing this value expands the search space, potentially improving recall by discovering more distant but relevant hubs. However, this also increases computational cost and runtime. The optimal value depends on the structure of the graph and the complexity of the question.

4.7.3.2. Retrieving Hubs at Current Level

The function getHubsAtCurrentLevel is described in Algorithm 15. It receives as input a list of entities $\vec{e}$ and a processed question $\hat{Q}$, and returns a list of relevant hubs together with the next set of entities to traverse. It first calls findHubRoots to identify candidate hub roots and the set of entities reachable in the next iteration level. If no hub roots are found, the function immediately returns with an empty list and the continuation set. Otherwise, the function retrieves up to `TopPathsToKeep` relevant `HubPath` objects for each hub root, by calling getHubPathsForHub. If no paths are returned for a particular hub, it is skipped. Otherwise, if valid paths are found, a new `Hub` object is instantiated and added to the result set. After processing all hub roots, the function returns the list of hubs together with the next entities to explore at the following graph traversal level.

Instead of skipping the hub, an alternative design could involve the invocation of the function buildHubs to run the indexing for the hub that did not return results. This function can also

Algorithm 15 Pseudocode for Retrieving Hubs at Current Level

Input:

$\vec{e}$: `List` of `EntityWithDirection` ▷ The entities to start the search from
$\hat{Q}$: `ProcessedQuestion` ▷ The processed question
`TopPathsToKeep`: `Integer` ▷ Number of paths to keep for this hub

```
1:  function getHubsAtCurrentLevel(ē, Q̂)
2:      (hub_roots, next_entities) ← findHubRoots(ē)
3:      if hub_roots = ∅ then
4:          return (∅, next_entities)
5:      hubs ← ∅                                          ▷ List of Hub objects
6:      for all root ∈ hub_roots do
7:          paths ← getHubPathsForHub(root, Q̂, TopPathsToKeep)
8:          if paths = ∅ then
9:              continue                                  ▷ No paths found for this hub
10:         hub ← Hub(root, paths)                        ▷ Create a new Hub object
11:         hubs ← hubs ∪ {hub}                           ▷ Add the hub to the list
12:     return (hubs, next_entities)
```

be used for each hub before retrieving the paths to ensure that they are up-to-date with the state of the graph. However, this would incur additional runtime during the retrieval process which is why this is not considered in the pseudocode.

4.7.4. Pruning Hubs

To prematurely filter out hubs that are irrelevant, the function pruneHubs which is illustrated in Algorithm 16 is used. This function is responsible for selecting the most relevant hubs from a list of candidates by calculating a weighted hub score for each hub and returning the top-ranked results. To calculate the overall score of a hub, the scores of the associated `HubPath` objects are aggregated. This is done by calculating the weighted average of the scores, where the weights are determined by an exponential function that emphasizes higher-scoring paths. Formally, let $\vec{s} = [s_1, s_2, \ldots, s_n]$ be the list of path scores associated with a hub. The weights are computed using the formula:

$$w_i = \exp(\alpha \cdot s_i)$$

The weighted hub score is then given by:

$$\text{Score of the Hub} = \frac{\sum_{i=1}^{n} w_i \cdot s_i}{\sum_{i=1}^{n} w_i}$$

Algorithm 16 Pseudocode for Pruning a List of Hubs

Input:

$\vec{H}$: `List` of `Hub` ▷ The hubs to prune

`PathWeightAlpha`: `Integer`

`HubsToKeep`: `Integer` ▷ Number of hubs to keep

Output:

A list of top-ranked `Hub` objects, sorted by the weighted average of their `HubPath` scores

function pruneHubs($\vec{e}, \hat{Q}$)
  for all $h \in \vec{H}$ **do**
    $scores \leftarrow \emptyset$ ▷ List of scores for each path
    for all $p \in h.Paths$ **do**
      $scores \leftarrow scores \cup \{p.Score\}$
    $weights \leftarrow \exp(\alpha \cdot scores)$ ▷ Exponentially weight the scores
    $weighted\_avg \leftarrow \frac{\sum(weights \cdot scores)}{\sum weights}$
    $hub.Score \leftarrow weighted\_avg$
  $hubs \leftarrow$ sortByScore($\vec{H}$) ▷ Sort hubs by their scores
  $relevant\_hubs \leftarrow \emptyset$
  for all $h \in hubs$ **do**
    if $|relevant\_hubs| \geq$ `HubsToKeep` **then**
      break ▷ Reached the limit of hubs to keep
    $relevant\_hubs \leftarrow relevant\_hubs \cup \{h\}$
  return $(relevant\_hubs)$

This weighting scheme amplifies the influence of higher scores, especially as the scaling factor α increases. Intuitively, it allows a few highly relevant paths to dominate the overall score of a hub, while suppressing the impact of lower-scoring paths. The detailed rationale behind this design decision is discussed in Section 4.9. After computing the weighted score for each hub, the list is sorted in descending order, and the top `HubsToKeep` hubs are selected. The result is a pruned list of the most semantically relevant hubs for the question.

4.8. Generation and Linking

After the `Hub` objects are collected, HubLink processes the content of those hubs to generate the answer to the question that is asked. The answer generation follows the same process regardless of the retrieval strategy applied. First, a partial answer is generated for each hub. This answer includes all the information that is relevant to answer the given question but is not required to provide a full answer. The full answer is generated in the next step through a synthesis of the partial answers. In addition, before the partial answers are created, the

Algorithm 17 Pseudocode for Partial Answer Generation

Input:
H: `Hub` ▷ The hub to generate a partial answer from
$\hat{Q}$: `ProcessedQuestion` ▷ The processed input question

Output:
A partial answer as a `String`, or `None` if no relevant answer is found

function getPartialAnswerForHub(H, $\hat{Q}$)
  $\hat{k} \leftarrow \emptyset$ ▷ Initialize additional knowledge
  $root \leftarrow H.Root$
  if $root.PathToEntity \neq \emptyset$ **then** ▷ Check if we have a path to a Topic entity
    $\vec{t} \leftarrow root.PathToEntity$ ▷ The path from the topic entity
    $\mathcal{T} \leftarrow$ pathToText($\vec{t}$) ▷ Convert the path to text representation
    $\hat{k} \leftarrow \hat{k} \cup \{\mathcal{T}\}$ ▷ Add the path to the additional knowledge
  $link\_knowledge \leftarrow$ getLinkKnowledge($H.Entity$, $\hat{Q}$)
  $\hat{k} \leftarrow \hat{k} \cup link\_knowledge$
  return queryLLM($\hat{Q}.Question$, $H.Paths$, $\hat{k}$)

linking procedure takes place. During this procedure, each hub is enriched with further information. In the subsequent sections, we will elaborate on each step in detail.

4.8.1. Partial Answer Generation

The generation of partial answers is an essential step that is done before the final answer is created. In addition to the data that has been retrieved from the graph, the HubLink approach further allows to enrich the context with data that is not stored directly in the graph. This is realized through the linking procedure, which is done before the generation of the partial answer. In this section, we are going to detail the pseudocode for both the partial answer generation and the linking procedure.

4.8.1.1. Prompting an LLM for Partial Answer Generation

The function `getPartialAnswerForHub`, shown in Algorithm 17, generates a partial answer based on the content of a given hub. It takes as input a `Hub` object H and a `ProcessedQuestion` $\hat{Q}$, and returns a string that represents a partial answer or `None` if no suitable answer can be generated from the hub.

Before querying the language model, the function gathers relevant contextual knowledge associated with the hub. First, if the *graph traversal retrieval* strategy is used, the root entity of the hub contains a known traversal path from a topic entity. This path can be helpful to understand the relationship between the topic entity and the hub. Consequently, this path is converted into a textual representation using the function pathToText and stored in a list to be later added to the prompt. Second, the function retrieves any available link

Algorithm 18 Pseudocode for the Linking of Hubs

Persistent State:
$\hat{D}$: `DataBase` ▷ External knowledge source

Input:
H: `Hub` ▷ The hub to collect additional knowledge for
$\hat{Q}$: `ProcessedQuestion` ▷ The processed user query

Output:
A `LinkData` object with additional information relevant to the hub

1: **function** GETLINKKNOWLEDGE($H, \hat{Q}$)
2: $\hat{k} \leftarrow \emptyset$ ▷ Initialize set of additional knowledge
3: $external\_knowledge \leftarrow \hat{D}$.GETDATA($H.Root.Entity$, $\hat{Q}$)
4: $\hat{k} \leftarrow \hat{k} \cup \{external\_knowledge\}$
5: **return** $\hat{k}$

knowledge for the hub that provides supplementary background information not directly encoded in the graph. This step is optional but can enrich the hub context, especially in cases where the graph is sparse or incomplete. This linking process is realized by calling the `getLinkKnowledge` function.

Once all relevant information is collected, the LLM is queried using the original question, the set of paths from the hub, and the additional knowledge $\hat{k}$. Concerning the details from the `HubPath` object that are shared with the LLM, both the text description and the triples are included. It is important that the LLM creates a partial answer that contains any potentially relevant information that could be useful to arrive at the final answer. If such a partial answer can not be created, the LLM should refrain from doing so. To achieve this, a carefully designed prompt, detailed in Appendix A.1.1, has been crafted.

4.8.1.2. Linking Procedure

Algorithm 18 presents the pseudocode for enriching a hub with external knowledge. The function GETLINKKNOWLEDGE receives a `Hub` object H and a `ProcessedQuestion` $\hat{Q}$, and returns a `LinkData` object containing additional information relevant to the hub.

In principle, any form of auxiliary knowledge that may enhance the quality of the answer can be linked to a hub. This procedure is based on the fact that the initial content of a hub consists only of data retrieved from the graph itself. By linking external sources, the context can be enriched beyond what is stored in the original graph structure, as further discussed in Section 4.9. The pseudocode illustrates one example of this enrichment process. It assumes an external database $\hat{D}$ that provides access to additional data. To retrieve this data, the identifier of the root associated with the hub is used. This can, for example, be the Digital Object Identifier (DOI) of a publication. The retrieved linking data is then returned as a `LinkData` object and is incorporated into the generation of the partial answer for the hub.

Algorithm 19 Pseudocode for Synthesizing the Final Answer

Input:

Q: `String` ▷ The original question
$\vec{S}$: `List` of `String` ▷ Partial answers from the hubs
$\vec{P}$: `List` of `HubPath` ▷ Paths associated with the hubs

Output:

A tuple (`retrieved_triples`, `final_answer`)

1: **function** getFinalAnswer($Q, \vec{S}, \vec{P}$)
2: $answer \leftarrow$ queryLLM($Q, \vec{S}$) ▷ Combine partial answers into a final response
3: $retrieved\_triples \leftarrow$ prepareOutputTriples($Q, answer, \vec{P}$)
4: **return** ($retrieved\_triples, answer$)

4.8.2. Final Answer Generation

The generation of the final answer is the last step of HubLink. This is done by synthesizing the partial answers into a final answer. In addition, the retriever also returns the triples that have been used for the answer generation to provide transparency and traceability. In this section, we are first going to detail the generation of the final answer and then describe how the triples that are returned are selected.

4.8.2.1. Prompting an LLM for Final Answer Generation

The function `getFinalAnswer`, shown in Algorithm 19, synthesizes the final response from the partial answers. It takes as input the original question Q, a list of partial answers $\vec{S}$, and the corresponding set of `HubPath` objects $\vec{P}$. The partial answers $\vec{S}$, along with the original question, are passed to a language model. The LLM is prompted to generate a final, coherent answer that integrates the evidence and insights contained in the partial answers. This approach allows the model to reconcile pieces of information and present a unified answer. The prompt used for this synthesis process can be tailored to any specific task. For example, the LLM can be directed to focus on identifying overlapping or conflicting parts. In the Appendix A.1.1, we provide a general prompt that is focused on the synthesis of information which may be further adapted. In the prompt that we provide, the answer is generated in a format specifically relevant for literature search, as each claim in the answer is tagged by the LLM with a citation mark $[i]$. Then, at the end of the answer, a list is appended that includes all the sources that have been referenced in the answer:

$$[i] \text{ Source}, [i+1] \text{ Source}, \ldots$$

After generating the final answer, the function calls prepareOutputTriples to extract and filter the most relevant triples from the original `HubPath` objects $\vec{P}$. This step identifies which triples contributed meaningfully to the answer, enabling the explainability and traceability of the result. Finally, both the synthesized final answer and the corresponding supporting triples are returned.

Algorithm 20 Pseudocode for Filtering the Output Triples

Input:

Q: `String` ▷ The original question
A: `String` ▷ The generated final answer
$\vec{P}$: `List` of `HubPath` ▷ Paths used during answer generation

Output:

A list of relevant `Triple` objects

```
1: function prepareOutputTriples(Q, A, P⃗)
2:     P⃗_sorted ← sortByScore(P⃗)
3:     triples ← ∅
4:     for all p ∈ P⃗_sorted do
5:         triples ← triples ∪ p.Path
6:     relevant_triples ← filterWithLLM(Q, A, triples)
7:     return relevant_triples
```

4.8.2.2. Filtering Relevant Triples

To make the answer of HubLink explainable and traceable, it returns a list of triples that were used in the answer generation. However, the hubs used in answer generation only provide a ranked list of paths. It is still unclear which of those paths actually contributed to the final answer. Algorithm 20 shows the pseudocode for identifying and returning the triples that serve as supporting evidence for a generated answer by filtering them from the paths.

The function prepareOutputTriples receives three inputs: the original question Q, the generated final answer A, and a list of `HubPath` objects $\vec{P}$ that were used in the reasoning process. The first step is to sort the hub paths based on their original relevance scores.This sorting allows the most relevant triples to be ranked at the top of the returned list. Next, all triples from the sorted paths are collected into a single set. To identify which of these triples are truly relevant to the final answer, the list is passed to a language model together with the original question and the generated answer. The LLM is tasked with selecting only those triples that directly contributed to or supported the final response. The resulting list of relevant triples is returned and used as the explainable output of the retrieval process. In Appendix A.1.1 we provide the prompt that is used for this filtering process.

4.9. Challenges and Design Rationale

During the development of the HubLink approach, we faced several challenges that influenced key design choices. In this section, we outline the reasoning behind these decisions. We begin by explaining our choice of an embedding-based retrieval approach. Next, we discuss the motivation for decomposing each `HubPath` into multiple levels. We then describe

the benefits of extracting components from the input question as a pre-processing step. Following that, we explain the role of a diversity ranker in enhancing the relevance of retrieved `HubPaths`. We also justify the use of weighted averaging when calculating the overall score of a hub. Additionally, we provide our rationale for incorporating a linking procedure and present the reasoning behind splitting the HubLink approach into two distinct strategies. Finally, we present the rationale behind the prompts that were engineered to realize the approach.

4.9.1. Using Embedding-based Retrieval

One significant challenge that we had to solve was the efficient retrieval of relevant graph data. One major reason is the exponential growth of candidate subgraphs as the graph size increases. This means that the number of graph elements that need to be considered during retrieval grows exponentially. This presents a substantial challenge because KGQA retrieval algorithms must ensure semantic relevance between the query and graph structures. This task is particularly difficult due to the high dimensionality of textual data within the nodes and edges of the graph. Consequently, this requires the development of heuristic search algorithms capable of efficiently exploring and retrieving relevant graph elements. [13, 120]

In addition, a suitable user experience is important. This requires balancing both the runtime and the costs of the approach.

We addressed the size of the graph by transforming the information of the graph into a vector space using a pre-trained embedding model. This enables the quick identification of relevant information through an ANN search between the stored vectors and the search query vector. By applying this technique, we reduced the complexity of the problem from having to deal with the size of the graph to dealing with how effectively we can query relevant vectors from the vector store.

4.9.2. Decomposing HubPaths to Multiple Vector Levels

It is challenging to consider both textual information and topological graph structures during graph retrieval. This is important because textual data convey information, while structural data describe the relationships between them. This requires the approach to account for both semantic information in the text and structural information in the graph. This challenge is further complicated by the fact that textual graphs often contain vast amounts of information, of which only a small portion is actually relevant to answering the query. Therefore, this problem requires the development of algorithms that can understand both textual and structural information and efficiently filter out irrelevant graph elements without losing critical information. [13, 120]

We encountered this challenge during the transfer of knowledge from the graph to the vector space as we needed to decide how to best store both textual and structural information. We

approached this issue by generating textual path descriptions with an LLM that captures both the information conveyed in the graph nodes and the relationships between them.

However, just converting the paths to textual descriptions and transforming them into dense vectors did not completely solve this issue. We found it to be important to consider the granularity of the vectors stored in the vector store. This is because irrelevant details can overshadow potentially relevant information. In other words, if too much information is loaded into a single vector at once, the similarity to the search query may decrease due to the presence of irrelevant data, causing relevant information to be missed. Conversely, if too little information is transferred into the vector, important details might be neglected, likewise reducing the similarity to the query. Thus, there exists a trade-off with respect to the amount of information embedded within a vector. Both an excess and a deficiency of information can result in relevant data not being found.

To address this trade-off, we decided on storing each `HubPath` in four different content levels, as detailed in Section 4.2.6. With this design decision, we reduce the risk that irrelevant data overshadows relevant data because there now exist multiple vectors, each representing a different level of content that can potentially match the question or its components. This design decision is closely related to the decision to extract components from the questions asked, which we discuss below.

Furthermore, we store the textual description and the triples of the path as metadata for each vector. This is important because the embedded text of the vector (other than the path level) loses context. To solve this, vectors are used for similarity matching during retrieval, and the answer is generated using the triples and path description of the associated `HubPath` to ensure enough context is included. A drawback of this approach is that each `HubPath` appears multiple times in the index (once for each content level). This can result in the same information being returned multiple times during retrieval without any added value. To prevent this, we apply a *deduplication* process that is realized by storing the unique hash value with each `HubPath` as additional metadata. During retrieval, we then ensure that for each hash value, only the top-scoring vector is considered.

4.9.3. Extracting Components from Questions

It is challenging to address the varying complexity of questions as they can range from simple to highly complex. Basic examples include: “Who are the authors of the paper?” while more complex queries might be: “Which reference architectures for cloud-native applications based on Kubernetes were published in 2020?” Complex questions consist of multiple aspects that need to be considered individually. During the design of HubLink, it became apparent that it is unlikely to find all relevant information for a complex question within a single path. As a result, doing the nearest-neighbor search using only the embedding of the question leads to insufficient results.

Our solution to this problem is to decompose the question into its individual components representing various semantic elements. Referring to the example question above, these components could be *reference architectures*, *cloud-native applications*, *Kubernetes*, and *2020*.

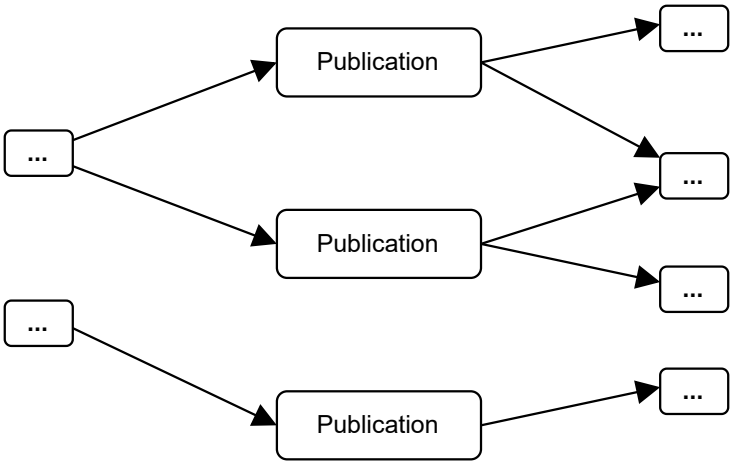


Figure 4.5.: Example structure of an RKG. Here, the information of interest is gathered as a subgraph around nodes of type *Publication*. HubLink uses these subgraphs by encoding them as Hubs to perform targeted inference.

Because we store the paths in the index at multiple granular levels, we increase the chances of achieving higher similarity scores using the extracted components. This is feasible because smaller vector units contain less information, enabling more accurate matching of specific content without reliance on the surrounding context.

4.9.4. Using Hubs to Semantically Store Information

When retrieving data from a graph, it is important to ensure that the relationships between information are retained correctly to avoid losing valuable data. This can be challenging, as it is not always clear how much of the surrounding information is actually needed [13, 120]. Our solution to this challenge is to decompose the graph into distinct `Hub` structures. This design choice is informed by the way information is organized within an RKG. As observed by Verma et al. [12], an RKG stores scientific data through research entities. Due to the inherent structure of graphs, related data tends to cluster around these entities. In this setup, the research entities act as roots, aggregating scientific information through their neighboring nodes.

Figure 4.5 illustrates this concept. It shows entities of the *Publication* type that inherently store scientific information about the publication around them. In this case, each publication entity serves as a root, with its associated data stored in the surrounding nodes. Given that questions in scholarly KGQA typically target information related to research entities, this natural clustering is exploited in the retrieval strategy of HubLink. We refer to this cohesive unit of information as a `Hub`, where each hub is based on a root node and encompasses all semantically related information accessible through paths originating from it.

Encoding these hubs in a vector space requires careful consideration of two key aspects: a) the significance of a node relative to the root node of the hub, and b) the preservation of relationships among the nodes within the hub. To meet both criteria, we consider all directed paths within the hub that start at the root and lead to end nodes. These paths are referred to as `HubPaths`. At the core, each `HubPath` represents a directed path within a hub that starts at the root entity of the hub, follows the direction of the graph, and either a) continues to the end of the graph, b) reaches a predetermined maximum length, or c) encounters a new root node of another hub. The idea of a `HubPath` is that the information on the path shares a common semantic meaning and should be encoded together in the vector space. Therefore, each `HubPath` is converted to vectors during indexing and stored in a vector database.

4.9.5. Applying a Diversity Ranker on the Triple Level

During development, we encountered an issue caused by retrieving triple-level vectors from the index. Since it is common that multiple triples (s, p, o) share the same subject s, this leads to a situation where many irrelevant triples receive a high similarity score simply because they share the same subject. This creates a problem during retrieval in which only a fixed number of `HubPaths` are selected per candidate hub.

To illustrate this, consider the following question: "Which authors does the paper with the title 'A Complexity Metric for Microservices Architecture Migration' have?". When this question is processed, it is decomposed into several components, one of which includes the paper title. Then, a similarity search is performed using these components. If the graph contains many triples where the title is the subject, these triples may dominate the top results due to the strong subject match, regardless of whether their predicates and objects are actually useful for answering the question. This effect can crowd out more relevant triples, such as those identifying the authors, from the top-k ranked `HubPaths` for a hub.

To mitigate this issue, we introduce a *Diversity Ranker*. This component penalizes triples whose subject has already been encountered, thereby encouraging a more diverse set of `HubPaths` in the results. This increases the chances of retrieving paths that contribute novel and useful information.

4.9.6. Using Weighted Averaging for Scoring of Hubs

In earlier stages of development, we calculated the score of a hub by taking the average of all associated `HubPath` objects. However, this approach revealed an issue that led us to adopt a weighted mean instead. Originally, the unweighted hub score was calculated as:

$$\text{Score}(Hub) = \frac{1}{n} \sum_{i=1}^{n} \text{Score}(h_i), \quad h_i \in \mathcal{H}$$

Here, $\mathcal{H}$ denotes the set of all `HubPath` objects for a given hub, $h_1, \ldots, h_n$ are the individual paths, and n is the number of these paths. The main drawback of this method is that all paths are treated equally, regardless of their individual relevance. As a result, a hub with many average-quality paths could end up with a higher total score than a hub that has fewer but highly relevant paths. This leads to a situation where important paths with high scores are overshadowed by a large number of less relevant ones, even though they might be critical for the overall quality of the hub. To address this issue, we now use a weighted mean, where paths with higher scores contribute more significantly to the overall score of the hub. Formally, let $\vec{s} = [s_1, s_2, \ldots, s_n]$ be the list of scores of all paths for a given hub. The weights for this new calculation are computed as follows:

$$w_i = \exp(\alpha \cdot s_i)$$

The final weighted score of the hub is then given by:

$$\text{Score}(Hub) = \frac{\sum_{i=1}^{n} w_i \cdot s_i}{\sum_{i=1}^{n} w_i}$$

The parameter α controls the extent to which high path scores are emphasized. Furthermore, this parameter can be fine-tuned to adapt to any given context.

4.9.7. Enriching Hubs by Linking to External Data

Although representing information as a graph introduces structure and emphasizes semantic relationships, it also comes with a trade-off: a loss of textual detail. This occurs because textual content must be adapted to the RDF format, which fragments information into triples, each capturing only a small portion of the original narrative. The goal of the linking process in HubLink is to complement the structured graph data with additional, richer context from external sources. This should enhance answer generation by providing more complete and informative responses.

To illustrate, consider a literature search scenario in which individual papers serve as the hub objects. Each hub is associated with a unique identifier, typically the DOI of the paper. This identifier enables the retrieval of additional information beyond what is captured in the graph. For example, we may connect to an external vector store that holds embeddings of the full-text from papers. Once relevant hub candidates are identified, the linking process uses their DOIs to perform targeted nearest-neighbor searches on these text embeddings. The returned text segments are then used to enrich the knowledge of the corresponding hubs. By combining structured graph data with context-rich textual extracts, this process can lead to more comprehensive and nuanced answers. In addition, it supports transparency in the retrieval process, as the enriched responses can be directly traced back to their source documents.

4.9.8. Providing Two Different Retrieval Strategies

During the development of HubLink, we decided to implement two different retrieval strategies. This decision was driven by a trade-off that emerged during development. By introducing both strategies, we allow users to choose the strategy that best suits their specific use case. In the following, we take a closer look at this trade-off.

The first approach, *direct retrieval*, enables fast retrieval using the vector index to quickly identify semantically similar content. Although this method is efficient and scalable, it becomes less effective in retrieving localized information as the size of the index grows. When multiple contexts from different sources share semantic similarity with the query, they may all be retrieved, regardless of whether they are relevant to the specific intent of the question. This can be problematic for questions that are meant to target particular segments of the knowledge graph.

To mitigate this limitation, traditional search systems often use metadata filters (e.g., research field, publisher, publication year) to narrow the scope of search results. HubLink incorporates a similar principle in its second strategy, *graph traversal retrieval*. In this approach, a specific entry point in the graph, referred to as a topic entity, is used to constrain the search to a relevant subregion of the graph. Without such a focus, the system would need to search the entire graph, increasing the risk of retrieving irrelevant information from unrelated areas. Consequently, the topic entity effectively acts as a filter, making this strategy particularly valuable in large-scale graphs containing millions of nodes.

However, using the *graph traversal retrieval* strategy comes with the drawback that a topic entity must be known or inferred. In some cases, such an entity may already be known. For example, if it is already known that a question refers to publications of a specific research field, the entity of that research field could be provided as input. Alternatively, the topic entity may be extracted automatically from the question, for example, through *named entity recognition* as demonstrated in [98]. Here, the terms provided in the question are mapped to entities in the graph using an LLM process.

Furthermore, another drawback of the *graph traversal retrieval* strategy is that traversing the graph is time-consuming and adds computational expense. This is because the hubs must be found by traversing the graph, which is a complex process that can take a considerable amount of time depending on the size of the graph and the number of hubs. The time requirement increases even further if the initial hubs do not provide an answer, and deeper levels must be traversed. Thus, the trade-off between the two retrieval strategies becomes increasingly relevant as the graph grows in size: it is a balance between the accuracy of the results and the computational cost and runtime. Consequently, we decided not to enforce a one-size-fits-all solution. Instead, users should choose the strategy that best fits their specific use case and requirements.

4.9.9. Prompt Engineering

The HubLink algorithm includes a total of four prompts, which are presented in Appendix A.1.1. During development, we observed that the performance of HubLink strongly depends on whether the prompts actually achieve the goal for which they are used. This is particularly true when generating partial answers. If the task description is too vague, too much irrelevant information will be included, and if it is too restrictive, too little information will be included. In the following, we briefly introduce our rationale behind each of the prompts in HubLink.

Partial Answer Generation Prompt This prompt is responsible for creating a partial answer for each hub. It is important here that even if the hub does not fully answer the posed question, all partial information provided by the hub for the given answer is formulated into a partial answer. We therefore created a prompt that prioritizes the integration of partial information even if it is not yet apparent whether the information is actually useful for the final answer later on. To support the LLM in understanding the task, we created a few-shot prompt that demonstrates how partial answers should be created from the hub information.

Final Answer Generation Prompt This prompt receives all created partial answers and generates a final answer by aggregating all partial answers. In addition, the LLM has the task of marking every fact in the answer with a source in the form of $[i]$. With this, we want to ensure that transparency about the information source is guaranteed during answer creation, which is particularly important in the scientific field. At the end of the generated answer, a list with the DOI and the title of the publication is appended.

Question Component Extraction Prompt This prompt is used to extract the components of the question from the given query. Here, we created a few-shot prompt to show the LLM the desired granularity for extraction.

Triple Filtering Prompt This is the last prompt in the execution. Here, based on the posed question, the answer, and the triples, a decision is made as to which triples are most relevant to the answer to filter out those triples that are not needed.

It should also be mentioned here that, depending on the application case, it may be reasonable to make adjustments to the current prompts. For example, during partial answer generation, the focus could be on identifying contradictions between sources instead of a simple synthesis of information.

4.10. Generalizability and Scalability

The fundamental design of HubLink offers potential regarding its range of application and performance capabilities. This section discusses these aspects. First, we explain why we consider HubLink to be schema-agnostic. Then, we examine the adaptability of the approach to other domains. Last, we explore the scalability of the approach by discussing how the modularity of the approach facilitates efficient processing even on KGs comprising millions of entities and relations.

4.10.1. Why HubLink is Schema-Agnostic

We define *schema-agnostic* as a characteristic of a system that does not rely on a fixed schema (i.e., a predefined set of classes, predicates, or relations). Instead, such a system operates on arbitrary or previously unseen graph structures without requiring adaptation of its core functionality.

HubLink is designed to be schema-agnostic because all core processing steps, including graph traversal, extraction, embedding, and retrieval, work on generic triples. Consequently, HubLink does not depend on any particular classes, property names, or specific ontology structures. This design makes the approach directly applicable to graphs with diverse or evolving schemas without necessitating changes to the underlying algorithms.

Nevertheless, the implementation of the isHubRoot classification function introduces a point where schema awareness can be incorporated, particularly if classification relies on entity types. The degree of sensitivity to schema evolution in such instances depends on the specific implementation chosen. For example, in the experimental setting conducted for this thesis, the *Paper* type from the ORKG served as a criterion for `HubRoot` classification. We argue that this specific choice does not render the overall approach inherently schema-reliant. This argument is supported by two main points: first, the designated type is fundamental to the graph structure in that context and is not anticipated to undergo frequent changes. Second, the construction and retrieval of paths within hubs remain entirely independent of schema-defined types. Furthermore, as defined in Section 4.2.3, the HubLink approach supports alternative methods for classifying `HubRoots` that do not use any direct schema information from the graph, thus offering a mechanism to maintain schema independence even in the hub classification stage.

4.10.2. Applicability to Other Domains

Although the current implementation of HubLink has focused on RKGs, which store scientific data, the approach is not limited to this domain. We believe that HubLink is broadly applicable to any field where the knowledge base is organized around well-defined entities. This includes domains such as internal company documents, Wikipedia articles, medical case records, or biological entities such as genes or proteins.

The key strength of HubLink lies in its ability to extract, aggregate, and retrieve information centered around specific objects. If these objects of interest are identifiable within the given data context, they can be used as hubs, making HubLink a flexible and adaptable solution across diverse domains.

4.10.3. Scalability to Large Graphs

In addition to cross-domain applicability, scalability is a critical factor for the practical use of HubLink. Although large-scale evaluations fall outside the scope of this thesis, we argue that HubLink is well suited for deployment on KGs containing millions of triples. This is because a unique advantage of HubLink lies in its modular, hub-based design. For queries that span multiple subgraphs or require aggregating information from distributed sources, the HubLink approach allows the graph to be decomposed into hubs. Each hub can then be individually evaluated to determine whether it contributes to answering the query. This allows the system to scale horizontally by distributing the evaluation across multiple hubs or even machines. This modular approach has the potential to handle complex queries efficiently by dividing the problem into smaller subproblems. We argue that HubLink opens up new possibilities for scalable and intelligent retrieval across large and complex KGs.

To realize the indexing of large-scale graphs, we propose two complementary strategies:

Indexing the entire Graph: One possible strategy is to index the entire graph using the embedding-based method of HubLink. Since the vector similarity search does not scale linearly with the size of the graph, response times remain relatively stable even as the graph grows. However, this property applies exclusively to the *direct retrieval* strategy because, in contrast, the *graph traversal* strategy requires evaluating all hubs that can be reached from the topic entity. However, this makes the traversal strategy particularly effective for queries targeting a specific region of the graph, as the topic entity naturally constrains the retrieval scope, leading to more focused searches.

Partitioning the Graph: An alternative approach that would also allow for control of the retrieval section involves partitioning the graph into multiple indices. This method proves advantageous when the relevant semantic domain of a query is known in advance. The underlying assumption is that large-scale KGs can be divided into semantically coherent subgraphs. Once such segmentation is performed, indexing can be limited to the subgraphs that are relevant to the queries. Consequently, it becomes necessary to select the appropriate index before processing a question. This selection can be carried out manually or automatically, for example, through an LLM-based classification mechanism.

4.11. Updating the Index

The HubLink retriever can only respond to questions based on the information stored in the associated index. For this reason, it is essential to update the index whenever changes are

made to the graph to ensure that the data remains consistent and up-to-date. The following section explains how this update process can be implemented.

As illustrated in the pseudocode of the function HUBINDEXNEEDSTOBEUPDATED, changes in individual hubs can be detected by comparing the hash values of the current `HubPath` objects with those stored in the index. A mismatch indicates a modification and thereby signals the need for an update. Alternatively, if each triple in the graph is annotated with a timestamp indicating its last modification, this metadata can also serve as a criterion for determining whether an update is required. When a hub is identified as outdated, it can then be refreshed by invoking the BUILDHUBS method. This procedure removes outdated entries from the index and rebuilds the corresponding `HubPaths`.

The pseudocode in Section 4.5 demonstrates how the index update can be implemented. This indexing function is designed to serve both initialization and maintenance purposes. During an update run, the algorithm iterates through each hub, verifying whether the paths stored in the graph remain consistent with those in the index. If discrepancies are detected, the affected components are updated accordingly. This procedure, which we refer to as the *Fixed Update* strategy, entails a comprehensive review of the index performed at specified intervals. Although computationally intensive, this approach guarantees that the entire index is synchronized upon completion.

An alternative strategy is the *Dynamic Update*, wherein updates are executed in real time in response to changes in the graph. When a modification occurs, the affected hubs are immediately updated using the BUILDHUBS method. This method requires the integration of a monitoring routine that is automatically triggered upon any update to the graph. Such a routine can either invoke BUILDHUBS directly or interact with the vector index to selectively adjust the relevant data entries.

In summary, we propose the following two strategies to maintain index consistency:

- **Fixed Update:** Involves a complete and periodic examination of the index to identify and refresh outdated hubs. This process is performed at predefined intervals and ensures full synchronization.
- **Dynamic Update:** Executes updates immediately following any modification to the graph. A monitoring routine initiates the update process, targeting only the affected components.

4.12. Limitations

The primary limitation of HubLink is the requirement of an index. This index is crucial for inference because only the information stored in the index is seen by the retriever. Consequently, if a question asks for information that is not indexed as part of a hub, it is not considered. This requires careful management of the index, for which we outline several possibilities in Section 4.11. Nevertheless, we argue that using an index instead of training

is a considerable benefit, as building the index itself does not require any additional data but the graph itself.

Beyond the limitation of the index, several other considerations affect the HubLink approach. The definition of optimal criteria to classify `HubRoot` entities from which the hubs are built is not straightforward and depends on the underlying graph. If these criteria are not well defined, relevant information may not be encapsulated within hubs, rendering it unreachable until the criteria and index are updated.

Furthermore, the indexing process itself, particularly for large or frequently updated graphs, can be computationally intensive and time-consuming. In addition, the decomposition of paths into multiple vector representations during indexing contributes to increased storage requirements of the index. These factors introduce maintenance overhead, as changes in the graph necessitate updates to the HubLink index.

Moreover, the answer generation capabilities of HubLink are significantly dependent on the LLM utilized. Whether the LLM is able to pre-filter extraneous information during the generation of partial answers and subsequently synthesize these into a coherent final answer relies upon the performance of the LLM and also whether the prompts are clear and specific enough for the LLM to understand. This may require the adaptation of prompts for different LLMs, tasks, or domains to achieve optimal results.

Operational aspects also introduce limitations. For instance, the graph traversal retrieval strategy requires a topic entity as an input, which implies either user provision or an auxiliary mechanism for entity linking from the query. Additionally, HubLink is an embedding-based system at its core and, as such, inherits certain limitations common to these types of systems. For example, as described by Wang et al. [98], the processing and interpretation of numerical constraints, such as dates, value ranges, or specific metric comparisons, presents challenges for such systems.

5. Taxonomy Construction Methodology

This section presents Contribution **C2.1**, a taxonomy construction methodology that aims to systematically compile knowledge from the literature to create an initial taxonomy, which is then improved and validated. The approach draws inspiration from the revised construction method by Usman et al. [80] by adapting the proposed *planning*, *identification and extraction*, *design and construction*, and *validation* phases.

The chapter is organized as follows. In Section 5.1 we first describe an evaluation plan that outlines how a taxonomy can be evaluated according to a GQM model. Then, in Section 5.2 our proposed taxonomy construction process is presented.

5.1. Evaluation Plan

This section includes an evaluation plan which is applied in the validation and application phases of our proposed taxonomy construction process. The plan is implemented following the GQM method [121, 122] and is illustrated in Table 5.1. The plan consists of two goals with a total of six questions and nine metrics. The metrics are based on the taxonomy evaluation approach proposed by [82] and are described in Section 2.7.3.

The first goal, **G1**, concentrates on validating the *suitability* of the taxonomy. It seeks to confirm that the taxonomy allows for the appropriate classification of the objects under study, maintaining the correct scope and level of granularity. Three questions address this goal. **Q1.1** evaluates whether the taxonomy achieves an appropriate level of generality and granularity. An insufficient number of classes may reduce informational utility, whereas an excessive number can introduce unnecessary complexity, hindering effective application. **Q1.2** assesses appropriateness, focusing on whether the taxonomy encompasses all necessary categories while excluding unnecessary ones. This involves balancing the inclusion of classifications required for comprehensive characterization against the avoidance of rarely utilized categories. **Q1.3** examines orthogonality, evaluating the degree of overlap among categories. Low orthogonality indicates redundancy due to overlapping categories. Minimizing such overlap enhances the precision of classification.

The second goal, **G2**, relates to the validation of the purpose of the taxonomy by assessing its quality and relevance relative to existing taxonomies. This goal encompasses three questions. **Q2.1** assesses the relevance of the classes and categories concerning the stated objectives of the taxonomy. This involves demonstrating the necessity of each element for fulfilling the intended function of the taxonomy. Question **Q2.2** investigates the novelty of the taxonomy,

Goal	Question	Metric
G1 Validate the suitability of the taxonomy by checking that it allows objects under study to be classified appropriately, with the correct scope and level of detail.	**Q1.1** Has the taxonomy reached an appropriate level of generality and granularity?	**M1.1.1** Laconicity **M2.1.2** Lucidity
	Q1.2 Is the taxonomy appropriate, comprising only necessary classes?	**M1.2.1** Completeness **M2.2.2** Soundness
	Q1.3 Is the taxonomy orthogonal without overlapping classes?	**M1.3.1** Orthogonality matrix
G2 Assess the quality and relevance of the taxonomy to existing ones to validate its purpose.	**Q2.1** Is the taxonomy relevant, comprising only necessary classes and categories?	**M2.1.1** Fraction of relevant classes and categories
	Q2.2 Is the taxonomy novel, having the right extent of new categories?	**M2.2.1** Innovation **M2.2.2** adaption
	Q2.3 Is the taxonomy significant which enables a more precise description?	**M2.3.1** Classification Delta

Table 5.1.: Overview of the GQM plan for the validation of the taxonomy.

specifically the extent to which it introduces new classes or adapts existing ones compared to established taxonomies. Finally, **Q2.3** evaluates the significance by determining whether the taxonomy facilitates a more precise description or classification within its domain than preceding taxonomies permit.

5.2. Development Process

The overall development process is illustrated in Figure 5.1. It consists of eight consecutive phases. The process begins with the Planning phase, where the intended use of the taxonomy is defined. In the following Literature Survey phase, a literature review is conducted to collect potential candidates that contain relevant terms, which we refer to as classes. These candidates are then manually extracted in the Extraction phase to identify the relevant classes from the text. In the Clustering phase, the extracted classes are merged in a deduplication process and subsequently grouped into semantic categories. These categories are then evaluated for their relevance in the Relevance Assessment phase. The remaining categories are used to initialize the first taxonomy increment during the Construction and Refinement phase. This increment is then evaluated in the Validation

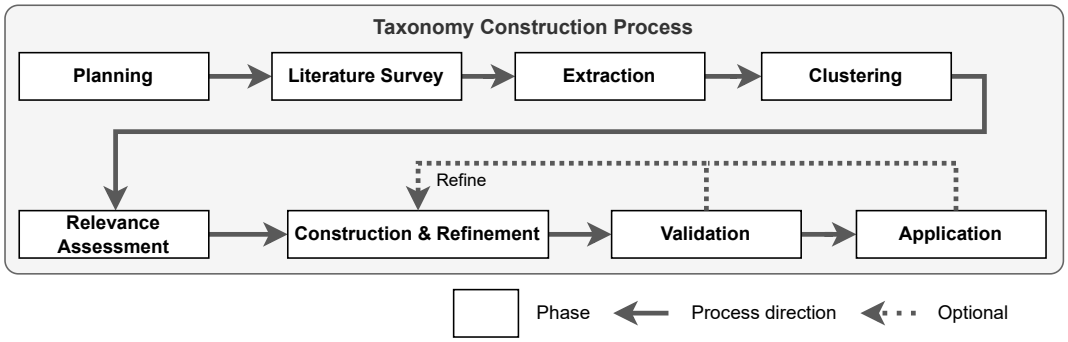


Figure 5.1.: The taxonomy construction process involving eight consecutive phases to synthesize knowledge from the literature into a taxonomy that is then incrementally refined and validated.

phase to determine whether revisions are necessary. If so, the process returns to the previous phase. Otherwise, it proceeds to the Application phase, where the practical use of the taxonomy is demonstrated in a specific application scenario. In the following sections, we take a closer look at each phase of the process.

5.2.1. Planning

The first phase of the process is Planning. In this phase, the objective and intended use of the taxonomy is defined. Additionally, characteristics of the taxonomy that should be considered during its development are specified.

According to Usman et al. [80], six activities should be carried out when planning a taxonomy. However, it is important to note that the authors focused specifically on the software engineering domain. For our process, we aim to maintain general applicability. Therefore, we have adapted these activities and formulated the following steps for the Planning phase:

S1: Define one or more declarative sentences that clearly define the objective and scope of the taxonomy.

S2: Define the domain for which the taxonomy is to be applied.

S3: Define the structure type of the taxonomy.

S4: Define the procedure type of the taxonomy.

S5: Define the topics and domains of the literature from which the collection of classes takes place.

In the first step, **S1**, the goal of the taxonomy should be clearly defined so that users understand the use cases it is meant to support. To achieve this, one or more declarative sentences should be formulated that narrow the scope and clearly express the intended purpose.

Step **S2** involves defining the domain of the taxonomy. This should help users assess whether the taxonomy is intended for a specific domain and allow them to better understand its scope.

In **S3**, the structure of the taxonomy is specified. According to Usman et al. [80], there are four different types of structures, which we will explain briefly. A taxonomy with a *hierarchy* structure has a single parent class that contains multiple subclasses. In contrast, a *tree* structure does not imply inheritance relationships between classes. Instead, it features a parent class with subclasses that can represent relationships such as part-whole, cause-effect, and process-product. A *paradigm* taxonomy represents a two-dimensional matrix classification, where each cell represents a combination of possible classes. Finally, a *faceted* taxonomy captures multiple independent perspectives or "facets", each with its own set of classes.

In step **S4**, the procedural type of the taxonomy has to be defined. This procedure describes how instances are systematically assigned to the classes or categories of the taxonomy. The procedure depends on the measurement method used for the classification and is divided into two types, according to Usman et al. [80]. *Qualitative* procedures use nominal scales, which means that the classification is based on descriptive attributes. *Quantitative* procedures, on the other hand, use numerical scales for classification.

The final step in the planning phase is **S5**, which involves categorizing the literature from which the classes will be extracted according to their topics and domains. This helps users understand the data foundation from which the taxonomy has been derived.

5.2.2. Literature Survey

The second phase of the development process is named Literature Survey. The goal is to identify candidates from the literature that potentially contain relevant classes for the taxonomy. The survey is conducted iteratively, with transparency and repeatability serving as top priorities. Furthermore, this iterative approach allows the process to be paused early if needed and resumed at a later time.

Two lists are defined to conduct the literature survey: *Intermediate List* and *Final List*. The intermediate list is defined for each iteration and is denoted as $\mathcal{L}_i$, where $i \in \mathbb{N}_{>0}$ indicates the current iteration. This list contains all the literature resources that have been processed during that specific iteration. This processing is done using the **Processing Paper Task** described below. During this task, new candidates may be extracted from a resource $\mathcal{P}$, which then forms the list for the next iteration, $\mathcal{L}_{i+1}$, such that:

$$\mathcal{L}_{i+1} = \bigcup_{\mathcal{P} \in \mathcal{L}_i} \left(\text{Ref}(\mathcal{P}) \cup \text{CitedBy}_n(\mathcal{P}) \right)$$

Here, $\text{Ref}(\mathcal{P})$ refers to the set of relevant references from the bibliography of $\mathcal{P}$, while $\text{CitedBy}_n(\mathcal{P})$ represents the first n publications assessed as relevant from a *Cited by* search.

In addition, the literature survey process has a second list named *final list* which is denoted as $\mathcal{F}$. This list contains all literature resources that have been assessed as relevant in any of the conducted iterations of the process.

The following section illustrates the literature survey process, which is based on the guidelines for conducting a systematic literature review by Wohlin [123]:

Literature Survey Process

1. **Prepare Seeds:** Collect seed papers and add them to $\mathcal{L}_1$.
2. **First Iteration:** Process each paper in $\mathcal{L}_1$ by applying the *Processing Paper Task* described below.
3. **Additional Search:** Conduct a search e.g. on Google Scholar to find additional papers that are not covered by the seed papers. Add these papers to $\mathcal{L}_2$.
4. **More Iterations:** Process each paper in the intermediate list of the current iteration $i \in \mathbb{N}_{>0}$ by applying the *Processing Paper Task* described below. Continue processing until satisfied or there are no more paper candidates in $\mathcal{L}_i$.

Processing Paper Task

1. Collect paper $\mathcal{P}$ from $\mathcal{L}_i$ that has not yet been processed.
2. Determine whether $\mathcal{P}$ is relevant on the basis of the title and abstract. This is done by applying the defined *inclusion* and *exclusion* criteria. If not relevant, $\mathcal{P}$ is skipped.
3. The text of $\mathcal{P}$ is skimmed to identify relevant text sections. If no relevant text sections are found, $\mathcal{P}$ is skipped.
4. If $\mathcal{P}$ is found to be relevant, it is added to the list $\mathcal{F}$.
5. The reference list of $\mathcal{P}$ is processed to identify possible relevant publications based on their title. Add these candidates to the $\mathcal{L}_{i+1}$.
6. Using a search engine like Google Scholar, the *cited by* feature is used on $\mathcal{P}$ to collect a list of publications $\hat{C}$ referencing $\mathcal{P}$. The list is obtained by sorting by relevance and collecting the first n publications. For each $\hat{\rfloor} \in \hat{C}$, their titles and abstracts are analyzed to consider whether the defined *inclusion* and *exclusion* criteria are met. If so, the paper $\hat{\rfloor}$ is added to $\mathcal{L}_{i+1}$.

To help categorize publications as relevant, inclusion and exclusion criteria should be defined. The exact criteria used depend on the respective objective and domain of the taxonomy.

5.2.3. Extraction

The result of the literature search is the set $\mathcal{F}$, which includes all publications that were classified as relevant. In the Extraction phase, each publication $f \in \mathcal{F}$ is processed by identifying classes for the taxonomy based on the content of the publication. For each relevant publication f, a subset C_f is defined, containing all classes extracted from f. Each class consists of a name and a description, both taken directly from the respective publication.

The complete set of all extracted classes is defined as the union of all C_f for every $f \in \mathcal{F}$:

$$C = \bigcup_{f \in \mathcal{F}} C_f$$

5.2.4. Clustering

The next phase is called Clustering and consists of two main steps. First, the classes C extracted from the literature are grouped to merge semantically redundant entries. Then, these consolidated classes are organized into categories.

Deduplication In the first step, the extracted classes C are clustered based on their semantic similarity. Classes that are considered equivalent or very similar in meaning, based on their names or descriptions, are grouped together. The level of abstraction at which this merging occurs depends on the specific use case.

Each resulting cluster contains classes that are either semantically equivalent or closely related. The set of these clusters is denoted by $\hat{C} = \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n$, where each cluster $\hat{c}_i$ is a subset of the original class set C, and all clusters are mutually disjoint:

$$\forall i \neq j : \hat{c}_i \cap \hat{c}_j = \emptyset \quad \text{and} \quad \bigcup_{i=1}^{n} \hat{c}_i = C$$

For each cluster $\hat{c}_i$, a representative class is defined, which provides the cluster with a name and description. This can be done either by selecting an existing class within the cluster or by manually synthesizing a new one based on the contained classes.

Categorization In the second step, the class clusters $\hat{C}$ are grouped into semantically coherent categories, denoted as $\mathcal{K}$. Depending on the context, it may be assumed that these categories are disjoint. Each category $k \in \mathcal{K}$ is assigned a name n_k and a description d_k. Naming is based on the interpretation of the deduplicated classes contained in $\hat{C}_k$, with the aim of capturing the overarching semantic context of the included concepts. The description d_k provides clarification and contextual background for the category. A category can thus be represented as a tuple:

$$k = (\hat{C}_k, n_k, d_k)$$

Here, $\hat{C}_k \subseteq \hat{C}$ is the associated group of deduplicated classes, n_k is the name of the category, and d_k is the corresponding description. The complete set of categories is then:

$$\mathcal{K} = \left\{ (\hat{C}_k, n_k, d_k) \;\middle|\; \hat{C}_k \subseteq \hat{C} \right\}$$

5.2.5. Relevance Assessment

During the Relevance Assessment phase, the goal is to evaluate the set of categories $\mathcal{K}$ and their corresponding deduplicated classes $\hat{C}_k$ in terms of their relevance to the target taxonomy. This step reduces the overall set $\mathcal{K}$ to only the categories and classes that are truly significant.

To assess relevance, an evaluation function is defined based on a specified set of qualitative criteria. Each category and class are manually evaluated through content analysis and reasoned judgment. The evaluation function is formalized as follows:

$$\text{rel} : \hat{C} \cup \mathcal{K} \rightarrow \{0, 1\}$$

Where:

1. $\text{rel}(x) = 1$ if the element $x \in \hat{C} \cup \mathcal{K}$ is considered relevant,
2. $\text{rel}(x) = 0$ otherwise.

Based on this evaluation function, the sets of relevant classes and categories are defined as follows:

$$C^* = \{\hat{c} \in \hat{C} \mid \text{rel}(\hat{c}) = 1\} \quad \text{and} \quad \mathcal{K}^* = \{k \in \mathcal{K} \mid \text{rel}(k) = 1\}$$

Furthermore, if a category $k \in \mathcal{K}$ is evaluated as not relevant, then all its associated classes $\hat{C}_k \subseteq \hat{C}$ are also classified as not relevant. Conversely, if a category does not contain any relevant classes, it can be excluded from $\mathcal{K}^*$.

5.2.6. Taxonomy Construction and Refinement

The phase Taxonomy Construction and Refinement may be carried out multiple times throughout the development process. The goal is to construct a taxonomy based on the categories and classes identified during the Relevance Assessment phase and to refine it iteratively as needed. Each taxonomy increment is denoted by $\mathcal{T}_i$, where $i \in \mathbb{N} > 0$. The first execution of this phase creates the initial increment $\mathcal{T}_1$. With each subsequent execution, the current increment $\mathcal{T}_i$ is revised to generate an improved increment $\mathcal{T}_{i+1}$.

Taxonomy Initialization To initialize the taxonomy, the set of categories rated relevant ($\mathcal{K}^*$) is used. This set forms the basis for the first taxonomy increment $\mathcal{T}_1$, which describes the structure and content of the taxonomy in its initial version:

$$\mathcal{T}_1 := \mathcal{K}^*$$

Taxonomy Refinement If it is determined during the Validation phase that the current increment $\mathcal{T}_i$ has weaknesses in terms of generality, appropriateness, orthogonality, or novelty, a targeted refinement step is carried out to address them. During this step, individual categories or classes may be created, adjusted, moved, merged, or removed. The result is a new increment $\mathcal{T}_{i+1}$, derived from a modified version of $\mathcal{T}_i$:

$$\mathcal{T}_{i+1} = \text{Refine}(\mathcal{T}_i, \Delta_i)$$

Here, Δ_i represents the set of all change operations identified during the refinement.

5.2.7. Validation

In the Validation phase, the current increment of the taxonomy $\mathcal{T}_i$ is evaluated based on the criteria *Generality*, *Appropriateness*, *Orthogonality*, and *Novelty*. For this validation, the goals **G1** and **G2** from the GQM plan are taken into account, which are outlined in Table 5.1.

Various metrics are used for validation, each having the form $f(\alpha, \beta)$. Here, α represents the set of classes from the current taxonomy, while β denotes the comparison set. In this context, $\mathcal{K}_{\mathcal{T}_i}$ refers to the categories of the current taxonomy, C_k to the not yet deduplicated classes within a category $k \in \mathcal{K}_{\mathcal{T}_i}$, and $C_{\mathcal{T}_i}$ to the complete set of classes in $\mathcal{T}_i$:

$$C_{\mathcal{T}_i} = \bigcup_{k \in \mathcal{K}_{\mathcal{T}_i}} C_k$$

As previously defined, C represents the set of all classes extracted from the literature. The metrics are thus applied as $f(C_{\mathcal{T}_i}, C)$, comparing the classes in the current taxonomy with the extracted set. Note that we refer to the classes before the deduplication process here. The validation is then carried out by answering the following questions according to the GQM plan:

Generality and Granularity Question **Q1.1** investigates whether the taxonomy demonstrates an appropriate level of generality and granularity. The metrics *Laconicity* (**M1.1.1**) and *Lucidity* (**M1.1.2**) are used for this evaluation. A high *Laconicity* score is expected in the first increment $\mathcal{T}_1$, since the initialization is based directly on the classes from the literature. In contrast, a low *Lucidity* score can be desirable depending on the context, as it indicates that many classes in the taxonomy are supported by multiple sources in the literature, which reflects broad acceptance.

Appropriateness Question **Q1.2** assesses whether the content of the taxonomy is suitable for its intended purpose. The supporting metrics are *Completeness* (**M1.2.1**) and *Soundness* (**M1.2.2**). A high *Completeness* score indicates that a large proportion of the extracted classes C are present in $\mathcal{T}_i$. In contrast, *Soundness* is at its maximum in the first increment $\mathcal{T}_1$, since all included classes at that stage are derived from the literature: $C_{\mathcal{T}_1} \subseteq C$.

Orthogonality Question **Q1.3** examines potential overlaps between classes of different categories within $\mathcal{T}_i$. An orthogonality matrix (**M1.3.1**) is created to make these overlaps visible.

Relevance Question **Q2.1** checks whether the classes included in the taxonomy are actually relevant to the intended purpose. The evaluation is based on a qualitative analysis for each class. After the relevance for each class has been asserted, a total relevance score (**M2.1.1**) is calculated. Since the initial taxonomy $\mathcal{T}_1$ is based on the pre-filtered set $\mathcal{K}^*$, a high level of relevance is already ensured. If new classes are added in later increments, their relevance must be discussed explicitly.

Novelty Question **Q2.2** addresses the degree of innovation in the taxonomy. It explores whether a meaningful number of new categories or classes have been introduced. The evaluation is based on the metrics **M2.2.1** (*Innovation*) and **M2.2.2** (*Adoption*). For the first increment $\mathcal{T}_1$, a minimal value is expected, as no new concepts have been introduced beyond those collected in $\hat{C}$.

If discrepancies are identified during validation, the process returns to the Taxonomy Construction and Refinement phase to create a new taxonomy increment $\mathcal{T}_{i+1}$ to resolve these issues.

5.2.8. Application

Once the Validation phase has confirmed that a taxonomy increment $\mathcal{T}_i$ meets the required quality standards, the process continues with the Application phase. The purpose of this phase is to demonstrate the applicability and usefulness of the taxonomy in specific scenarios.

To accomplish this, the final increment $\mathcal{T}_i$ is applied to selected use cases. This involves assigning concrete questions, objects, or entities to the categories and classes of the taxonomy. If issues arise during this process, such as missing, unclear, or unsuitable categories or classes, the development returns to the Taxonomy Construction and Refinement phase. In this case, a new increment $\mathcal{T}_{i+1}$ is created to address the identified issues. Therefore, the Application phase not only serves to demonstrate the practical utility of the taxonomy but also acts as a final validation loop in the form of a controlled real-world test.

The application phase is further supported by question **Q2.3**, which is evaluated using the metric *Classification Delta* (**M2.3.1**). The goal is to determine the significance of the taxonomy in terms of its actual classification performance. Formally, let $\mathcal{U}$ be a set of use

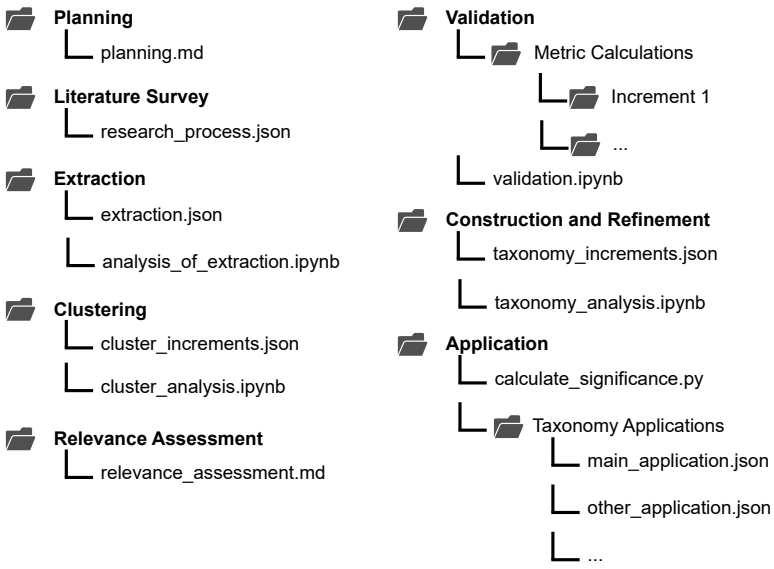


Figure 5.2.: Provided artifacts to support the taxonomy construction.

cases to be tested, and let $\hat{\mathcal{T}}$ be a comparison set of alternative taxonomies. The metric calculates the classification difference as follows:

$$\Delta(\mathcal{U}, \mathcal{T}_i, \hat{\mathcal{T}}) = f_{\text{delta}}(\mathcal{U}, \mathcal{T}_i, \hat{\mathcal{T}})$$

The higher the value of Δ, the more significant the evaluated taxonomy is compared to the alternatives. However, the acceptable level of significance depends on the specific use case and must be interpreted appropriately within the domain-specific context.

5.3. Supporting Construction Artifacts

To facilitate the practical implementation of the taxonomy construction process, we provide a set of supporting artifacts, illustrated in Figure 5.2, which are available in our replication package [124].

The provided artifacts are organized according to the corresponding phases of the construction process. For each phase, practitioners can access structured JSON files and Jupyter Notebooks that enable systematic execution of the process steps, comprehensive analysis of results, and calculation of relevant metrics. In addition, a usage guide is included in the replication package, offering instructions for leveraging these artifacts effectively throughout the taxonomy construction and evaluation workflow.

5.4. Limitations

In this section, we acknowledge several limitations of our proposed taxonomy construction approach.

First, several phases involve manual effort and qualitative judgments, which can introduce subjectivity and potentially affect reproducibility. For example, the extraction of classes, the deduplication and categorization steps within the Clustering phase, and the Relevance Assessment depend on human interpretation and the definition of the guiding criteria. The iterative nature of refinement also relies on judgments regarding when a satisfactory state is achieved, which can influence the overall duration and outcome of the process.

Second, the effectiveness of the Application phase as a final validation step is dependent on the selection and diversity of the use cases. If the chosen scenarios do not adequately represent the intended scope of the taxonomy, some of its deficiencies might not be identified. As a consequence, the generalizability of the validation results from this phase is tied to the representativeness of these application instances.

Finally, the initialization of the taxonomy is based on a literature survey. This foundation means that the process is dependent on the availability, quality, and scope of the existing literature. The comprehensiveness and representativeness of the resulting taxonomy are therefore contingent upon the body of literature identified and processed. For example, if the literature in a specific domain is sparse, exhibits bias, or is of inconsistent quality, this will inevitably impact the foundation upon which the taxonomy is built. Furthermore, the selection of seed publications and the execution of searches using Google Scholar may introduce selection bias if not carefully considered and justified.

6. KGQA Retrieval Taxonomy

This chapter presents the second primary contribution, **C2**, of this thesis: a taxonomy for the classification of questions relevant to KGQA systems, specifically within the context of literature search tasks. The taxonomy categorizes questions based on characteristics such as operational requirements and structural complexity, which are key dimensions influencing the retrieval performance of KGQA systems. Consequently, the taxonomy supports the generation of diverse question datasets, thereby facilitating a thorough assessment of KGQA system retrieval capabilities across various question types.

The development of this taxonomy followed the construction process detailed in Chapter 5. Accordingly, this chapter is structured to mirror the distinct phases of that methodology. Section 6.1 begins by outlining the Planning phase, defining the objectives and intended applications of the taxonomy. The subsequent Literature Survey phase, documented in Section 6.2, describes the identification and selection of relevant literature candidates. Section 6.3 then details the Extraction of classes from these candidates, which are subsequently grouped during the Clustering phase presented in Section 6.4. Following this, Section 6.5 documents the Relevance Assessment of these clusters. Based on this assessment, the initial taxonomy is constructed and refined as described in Section 6.6. The resultant KGQA retrieval taxonomy is presented in its final form in Section 6.7. Its practical utility is then demonstrated through an application in a SWA context in Section 6.8. Concluding the chapter, Section 6.9 discusses the potential threats to validity associated with the construction of this specific taxonomy.

6.1. Planning

According to the construction approach, the planning phase should include five different steps that guide the objective and intended use of the proposed taxonomy (see Section 5.2.1). In the following, we will describe each of the individual steps.

S1: The main objective of the taxonomy is to *classify the characteristics of questions posed to KGQA systems for scholarly literature search tasks to facilitate the creation of diverse sets of questions that test a broad range of retrieval capabilities.*

S2: The taxonomy is designed to be utilized specifically within the domain of scholarly data that is stored within an *RKG*.

S3: The classification structure type selected for the taxonomy is a *facet-based* structure. It defines multiple distinct categories, where each has its own set of classes. The advantage of such a structure is that it can easily be adapted to changes in the classification requirements.

S4: The classification procedure type is *qualitative* since it is based on nominal scales.

S5: The taxonomy consists of question classification types drawn from the literature. For the literature survey, we have defined the following categories as relevant: The first category is *Research Questions*. In this category, the literature provides information on questions relevant to the research process of a practitioner. The second category is *Question Classifier* since it includes literature that provides a model or approach to classify questions according to a taxonomy. As these publications are likely to provide information about the types that the classifier is classifying, this category is expected to provide valuable insights. Finally, the *Knowledge Graph Question Answering Dataset* category includes literature proposing a QA dataset specifically for KGs. This is particularly useful for our use case as it has the potential to provide KG-specific information.

6.2. Literature Survey

In this section, we document the literature survey that was conducted as part of the taxonomy development. A total of two iterations were performed to collect relevant candidate papers.

In the following, we start by defining the inclusion and exclusion criteria that have been specified to guide the assessment of relevance for paper candidates. Then, we describe the first iteration of the literature survey. Next, we outline the Google Scholar search that was carried out to find additional paper candidates. Finally, we detail the second iteration of the literature survey.

6.2.0.1. Inclusion and Exclusion Criteria

Following the advice of the construction approach in Section 5.2.2, we define the following *inclusion* and *exclusion* criteria to assess the relevance of papers during the survey:

Inclusion Criteria

- Publications that directly or indirectly propose classes for the classification of questions.

Exclusion Criteria

- Publications for which the full text is not available.
- Publications that do not contain information on classes for the classification of questions.
- Publications that are not written in English.

In the following, we outline the two search iterations that were carried out. The search was stopped after two iterations in order to keep the scope of the thesis manageable. We also believe that, by the end of the second iteration, we had exhausted the pool of interesting information reachable from the seed papers and the applied search queries. This is supported by the fact that only 22 candidates are considered for a third iteration, which is noticeably lower than the 46 candidates examined during the second iteration. Therefore, if future work is interested in the continuation of the research, we recommend adding more search queries to increase the number of candidates. This continuation can be carried out seamlessly by utilizing our prepared artifacts in the replication package [124].

6.2.1. First Literature Survey Iteration

We populated the initial *intermediate list* $\mathcal{L}_1$ with 11 publications provided by the thesis supervisor. After reviewing each paper, five publications have been excluded for not meeting the inclusion criteria or for not complying with the exclusion criteria. Through processing of the references and citations of the remaining papers, we added 19 papers to $\mathcal{L}_2$ and a total of six publications to the *final list* $\mathcal{F}$.

Looking at the relevant publications, some focus on the creation of QA classifiers. Li and Roth [32] define a two-layered hierarchical taxonomy of question types in an open-domain context and learn a hierarchical question classifier based on this taxonomy. In addition, Singhal et al. [33] present a QA system based on a taxonomy of question types.

Instead of creating QA classifiers, other publications focus on the creation of KGQA datasets. Auer et al. [104] present a scientific KGQA dataset that has been created based on the ORKG. The dataset focuses on scholarly questions and includes a variety of types for the classification of questions. Furthermore, Karras et al. [37] present a dataset of competency questions for the ORKG. Although they do not provide a question taxonomy explicitly in their paper, we extracted the question types from these instantiated questions.

Specifically in the domain of SE, Shaw [125] describes key components and considerations necessary to write effective research articles, which also include classes specifically suited for research questions. Furthermore, Easterbrook et al. [126, pp. 287–290] provide a comprehensive guide for selecting empirical research methods in SE. Their work includes a categorization of the kinds of research question asked in SE.

6.2.2. Search for Additional Relevant Paper Candidates

To find additional sources of information and avoid clustering around a single topic, we conducted a search on Google Scholar after the first iteration. Our intention with this search was to find papers that were not already covered by the seed papers, meaning that they have not already been added in the lists $\mathcal{L}_1$ or $\mathcal{F}$. The search was carried out using the following search queries, which we chose based on the objective that we defined in the planning phase:

Q1: "research questions" AND "classification" OR "taxonomy" OR "types"

Q2: "questions" AND "classification" OR "taxonomy" OR "types"

Q3: "research questions" AND "construction" OR "development" OR "formulation"

Q4: "question" AND "construction" OR "development" OR "formulation"

Q5: "software engineering" AND "research questions" OR "classification" OR "taxonomy"

Q6: "question answering" AND "classification" OR "taxonomy" OR "types"

Q7: "question answering" AND "datasets" AND "graph"

Q8: "question answering" AND "scholarly" AND "graph"

Q9: "question answering" AND "structure"

For each query, we looked at the first 40 results that have been returned sorted by relevance. The full search process that details each paper that was added by the search queries is available in our replication package [124]. In the following, we provide an overview of the query results.

The first query **Q1** returned 3,620,000 results, from which we added four new candidates to $\mathcal{L}_2$. **Q2** returned 6,560,000 results. Although some publications looked interesting on the basis of their titles and abstracts, their full texts were not available. The next query **Q3** had 2,960,000 results from which we added four more candidates. Query **Q4** returned 4,560,000 results, but we excluded each paper through the application of our inclusion and exclusion criteria. Next, **Q5** returned 1,240,000 results, from which we added five publications to $\mathcal{L}_2$. In addition, one paper was found through the query that was already added to $\mathcal{L}_1$. The following query **Q6** returned 308,000 results, from which one paper was already considered in our candidates, and four more papers were added. The next query **Q7** had 76,900 results from which one paper was already considered and three more were added. Query **Q8** had the least number of results, with 5,040 papers returned, but the most relevant papers to our cause. We added three more papers to our candidates for the next iteration. Eight papers that the query returned were already in $\mathcal{L}_1$, $\mathcal{L}_2$ or $\mathcal{F}$. Finally, **Q9** had 196,000 results, from which we added four more candidates to our list. One of the returned papers was already on our list. In summary, a total of 28 new publications were added to $\mathcal{L}_2$ based on these searches.

6.2.3. Second Literature Survey Iteration

The second iteration started with a total of 47 papers in $\mathcal{L}_2$. After processing each paper, we added 23 publications to $\mathcal{L}_3$ for the next iteration. Furthermore, we added 21 papers to the final list $\mathcal{F}$.

Several of the new papers added to $\mathcal{F}$ propose new datasets or benchmarks for KGQA. Dubey et al. [31] introduce LC-QuAD, which is a large dataset of natural language questions with paraphrases along with corresponding SPARQL queries for Wikidata and DBpedia. In their work, they specifically provide a taxonomy of classes for question classification. In addition, Bordes et al. [114] present the SimpleQuestions dataset, which is specifically designed for single-fact retrieval. Although they provide minimal information on question types in their work, we were able to identify some classes. Tran et al. [127] developed COVID-KGQA, which is a multilingual corpus related to COVID. In their work, they also evaluated the influence of a "Wh"-based question taxonomy on the performance of their dataset. Usbeck et al. [118] add another KGQA dataset with QALD-10, which is a QA benchmarking dataset. They provide some information about the types of questions that are included in the dataset. Furthermore, Banerjee et al. [24] created the DBLP-Quad dataset which is focused on scholarly questions. Their work also provides a taxonomy that describes the types of question that the dataset consists of. Another dataset is Hybrid-SQuAD, which has been designed to address scholarly QA by Taffa et al. [128]. In addition, Jaradeh, Stocker, and Auer [28] present a BERT-based QA system for tabular representations in ORKG, together with the ORKG-QA benchmark, a small dataset of scholarly questions. Steinmetz and Sattler [129] survey the challenges for KGQA datasets and extract the answer types from these.

Beyond the datasets themselves, several studies focus on creating classifier models with the task of classifying questions. Bolotova et al. [130] provide a systematic categorization of non-factoid questions and their corresponding answer structures intended for open-domain QA systems. Liu and Jansen [131] similarly propose a taxonomy for classifying questions on social networks such as Twitter. Furthermore, Moldovan et al. [105] provide a comprehensive taxonomy for open-domain QA that integrates both syntactic and semantic techniques, while Riloff and Thelen [132] discuss a rule-based system for reading comprehension and illustrate how hand-crafted rules depend on distinct question types. Furthermore, Nguyen and Pham [133] propose an ontology-based QA system for the Vietnamese language, including question structures as part of their work. Chernov, Petukhova, and Klakow [134] describe a question interpretation module for a dialogue-based quiz QA application.

Another set of publications examines how to classify or formulate research questions. Dillon [106] investigates the classification of research questions as suggested by Aristotle. Ratan, Anand, and Ratan [135] emphasize how a well-structured research question facilitates effective scholarly work. In addition, Kamper [136] discusses the importance of well-defined research questions in the field of healthcare. They provide and discuss a small taxonomy of question types. Sjoberg, Dyba, and Jorgensen [137] propose a taxonomy of archetype classes for SE research, suggesting a foundational framework for shaping and guiding empirical studies.

Category	Domain	Classes	Papers
KGQA Dataset	General	27	4
KGQA Dataset	Requirements Engineering	3	1
KGQA Dataset	Scholarly	38	4
KGQA Dataset	Covid	9	1
Question Classifier	General	58	6
Question Classifier	Social Science	4	1
Question Classifier	Spoken NLP	8	1
Question Classifier	Vietnamese Language	22	1
Research Questions	General	12	2
Research Questions	Design Science	10	1
Research Questions	Software Engineering	21	2
Research Questions	Healthcare	3	1
Other	General	8	1
Other	Software Engineering	4	1
Total		227	27

Table 6.1.: Distribution of the extracted classes by the papers they were extracted from.

Other papers focus on the design and evaluation of QA systems. Allam and Haggag [138] provide an overarching view of QA and its core components, revealing how the types of questions factor into the architecture of the system. A more conceptual approach is seen in Mikhailian, Dalmas, and Pinchuk [139], who introduce the ideas of Asking Point and Expected Answer Type as guiding concepts to capture the intent of the question.

6.3. Extraction of Classes

A total of 81 publications were considered in our literature survey in the first ($\mathcal{L}_1$) and second ($\mathcal{L}_2$) iterations. Among these publications, 27 were selected as sources for class extraction ($\mathcal{F}$). After the search, we performed a manual extraction of the classes. A total of 227 classes were extracted from these publications.

In the following sections, we analyze the distribution of the classes and publications from which the classes have been extracted. We first look at the distribution of the classes by the category and domain of the publication. Subsequently, we analyze the publication years prior to evaluating the citations and references among the papers.

6.3.1. Distribution of Classes by Category and Domain

During the search, we assigned each paper to one of the categories introduced in the planning step. If a paper could not be clearly classified into any of the predefined categories,

we added it to the category *Other*. Furthermore, each publication was assigned a domain to understand its scope.

Table 6.1 shows the distribution of the publications and extracted classes based on these categories and domains. The table illustrates that most of the question classes come from the category *Question Classifier*, which includes those publications that propose methods to classify questions. This category comprises 92 question classes sourced from nine publications in four domains. Most of the papers in this category fall into the General domain, which means that their classifications are not limited to a specific topic area. Each of the remaining domains in this category (Social Science, Spoken NLP, Vietnamese Language) is represented by only a single paper. It is logical that this category includes the largest number of classes, as considering question structure and types is crucial when building classifiers. The classes extracted from these papers primarily relate to a form of answer type in which the expected format and content of the answer are considered, and the types of questions, which reflects the data processing required to derive an answer.

The second largest category by number of extracted classes is the *KGQA Dataset* category (77 classes from 10 papers). This category includes papers that propose a dataset for KGQA and is represented by four domains, where the General domain and the Scholarly domain are equally represented by four papers each. The other two domains covered are Requirements Engineering, and one paper related to Covid. In general, this category provides critical insights relevant when working with KGs. The literature suggests considering the structure of the KG when classifying questions, which can indicate the complexity of the retrieval task. Additionally, considering the required KG operations is relevant for understanding processing needs.

The third most common category is *Research Questions*, which includes publications focused on the formulation and structure of research questions. Represented by six papers, this category includes 46 extracted question classes from four different domains. The General and Software Engineering domains are equally represented, each with two papers. The other two domains are related to Design Science and Healthcare. Generally, this category is useful for understanding the needs of researchers, which is particularly important for our taxonomy. As such, the literature highlights the importance of considering the focus of a question and its intended goal.

Finally, the *Other* category includes publications that did not fit the predefined categories. This category is represented by two papers and includes 12 extracted question classes. The domains of these papers are Software Engineering and the General domain.

6.3.2. Distribution of Publications by Year

Figure 6.1 shows the distribution by publication year for the papers from which we extracted classes. The years range from 1984 to 2024, with most publications published in the last decade. The oldest paper is authored by Dillon [106], who investigated the classification of research questions as suggested by Aristotle. The most recent paper is authored by Taffa et al. [128], which presents the Hybrid-SQuAD dataset designed to address scholarly QA by

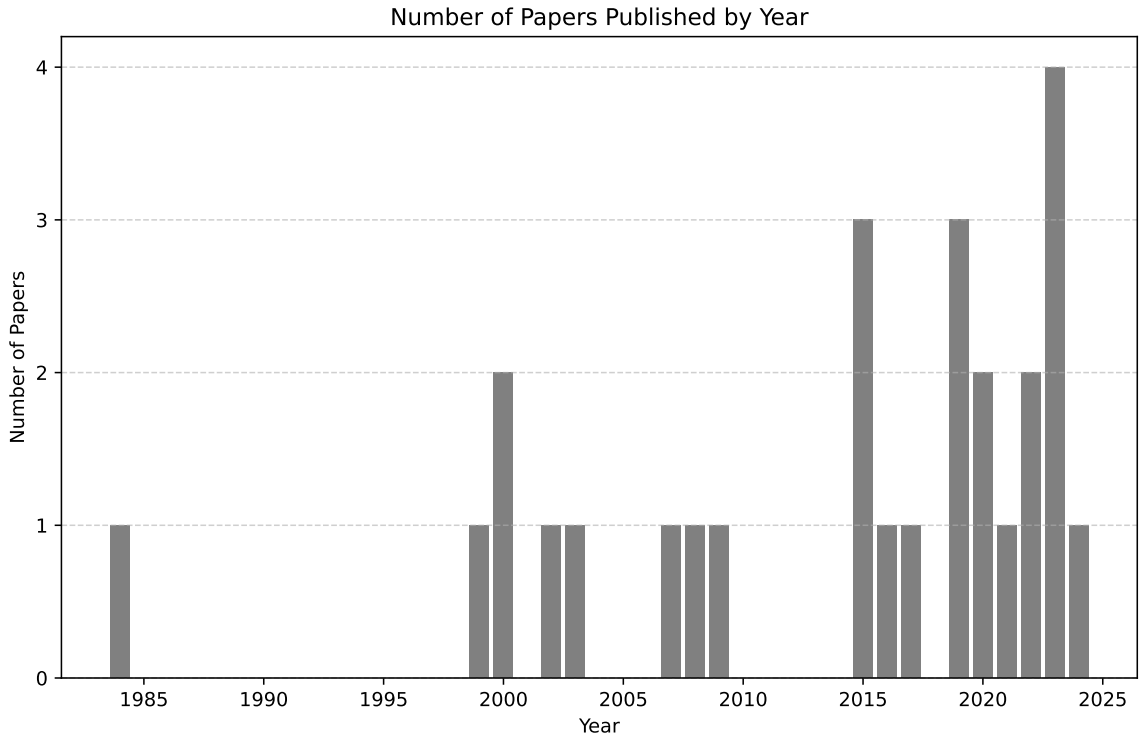


Figure 6.1.: The distribution of papers by their year of publication.

combining tabular and textual data. The year 2023 yielded the most publications in our set (four papers), while other years contributed between one and three papers. Several years within the range had no publications matching our criteria.

6.3.3. Distribution of Citations and References

Table 6.2 illustrates the frequency with which the papers in our selected set ($\mathcal{F}$) cite each other. This provides insight into the relative influence of each publication within our selection. In particular, [32] is the most frequently cited paper, indicating a potentially greater influence within our selected set of papers. In addition, five papers are cited twice and five are cited once, suggesting a modest influence within the set. Of the 27 papers in the final list, 16 were not cited by any other paper within this set, suggesting that they have had limited direct influence on the classification schemes developed in the cited works within our collection.

In contrast, Table 6.3 details how often each paper references other publications within our selection. This data reveals the extent to which the classifications may have been informed by prior work. According to the data, [104] stands out with six references, indicating that its classifications are heavily influenced by other publications on our list. Similarly, [134] references four papers, while both [37] and [128] cite three publications each. In addition, four papers [130, 138, 24, 118] include only one reference. Overall, with only eight out of 27

Paper	Number of Citations
[32]	[104], [130], [138], [134]
[31]	[104], [118], [24]
[33]	[104], [134]
[139]	[104], [134]
[104]	[37], [128]
[28]	[37], [128]
[114]	[104]
[132]	[104]
[137]	[37]
[105]	[134]
[24]	[128]

Table 6.2.: The number of times a paper has been cited by another paper within $\mathcal{F}$.

Paper	References
[104]	[114], [31], [32], [33], [132], [139]
[134]	[33], [105], [139]
[37]	[137], [104], [28]
[128]	[104], [24], [28]
[130]	[32]
[138]	[32]
[24]	[31]
[118]	[31]

Table 6.3.: The number of times a paper has referenced another paper within $\mathcal{F}$.

papers referencing other works in the data, it appears that most of the publications derive their insights independently.

6.4. Clustering of Extracted Classes

After extracting 227 classes for question classification, we performed the Clustering phase of the taxonomy construction process. The goal of the clustering process is to semantically group classes with similar names or descriptions and to merge duplicate classes proposed by multiple publications, thereby providing an overview of the unique classes and reducing redundancy. The clustering process is carried out in two steps: *Deduplication* and *Categorization*, as described in Section 5.2.4.

6.4.1. Deduplication Process

In the first step, the names of the question classification classes and the descriptions provided by the respective authors were manually analyzed for similarities. All classes C that shared a similar or identical name and had semantically similar descriptions were grouped together. After performing this process, the 227 classes in C were consolidated into 96 merged classes $\hat{C}$. Each merged class was then given a unique name and description by combining the names and descriptions of the classes within it.

6.4.2. Categorization Process

In the second step, we clustered the classes $\hat{C}$ into categories. For this purpose, we analyzed the names and descriptions of the groups to identify commonalities. Classes that exhibited coherence were grouped into a category, resulting in a total of nine categories. Finally, based on the combined names and descriptions of the classes $\hat{C}$ within the categories, we established a unique name and description for each of the categories. The detailed process is documented in our replication package [124]. An overview of each category and its metadata is shown in Table 6.4. In the following, we present the final categories with their names, descriptions, and characteristics.

Category 1 We identified that the classes in this category are about the representation of knowledge in a KG. It describes the granularity of the facts that the retriever needs to retrieve in order to arrive at the answer. The cluster distinguishes between two classes. First, questions that require the retrieval of multiple facts from the graph to be answered. Second, questions that require only one fact. As such, we named this category *Graph Representation*. Five sources form this cluster [24, 104, 31, 28, 114]. From the categories of these papers we can conclude that they originate from the development of KGQA datasets, with most contributions emerging in the last five years. Moreover, the contributions span both the academic and general domains, suggesting a broad applicability of this distinction.

Category 2 This category contains the largest number of classes among all categories, encompassing a total of 33 classes derived from 25 different publication sources [138, 105, 129, 33, 106, 132, 128, 139, 137, 133, 134, 32, 125, 130, 24, 104, 31, 28, 127, 126, 135, 136, 131, 37, 107]. Each class categorizes a question based on the expected format and content of its answer. These classes span a wide range, including *entities*, *names*, *dates*, *monetary values*, and *bibliometric numbers*. We find that some classes are addressed by multiple sources. The *boolean* class is the most widely considered, appearing in 10 sources. This is followed by the Human/Person, Location, and Quantitative classes, each appearing in seven sources. The classes *date* and *description* are also commonly represented, with five sources each. Additionally, the classes *undefined*, *time*, *organization*, and *entity* are mentioned in four sources each. The remaining classes are referenced less frequently. Overall, we named this category *answer type*.

Category 3 This category consists of classes that define the types of operations a retriever must perform to arrive at an answer, such as *negation*, *contingencies*, *counting*, and *comparison*. With contributions from 17 different sources [138, 137, 24, 104, 133, 126, 106, 135, 136, 31, 118, 129, 127, 130, 28, 114, 37], it is the second most prevalent cluster identified in our extraction. The majority of these sources stem from the KGQA dataset literature, although the category is also addressed in works on question classification and in studies focused on scholarly research questions. It is primarily considered within open-domain and academic contexts but also receives attention in the software engineering domain. In addition, specialized fields such as requirements engineering and Covid-related research contribute to this category. We refer to this category as *Question Types*.

Category Name	Classes	Sources	Domains	Years	Source Categories
Graph Representation	2	5	Scholarly, Open	2015, 2019, 2020, 2023	KGQA Dataset
Answer Type	33	25	Scholarly, Open, Software Engineering, Language specific, Spoken NLP, Covid, Healthcare, Social Science, Requirements Engineering, Design Science	1984, 1999, 2000, 2002, 2003, 2007, 2008, 2009, 2015, 2016, 2017, 2019, 2020, 2021, 2022, 2023, 2024	Question Classifier, KGQA Dataset, Research Questions, Other
Question Types	18	17	Scholarly, Open, Software Engineering, Language specific, Healthcare, Covid, Requirements Engineering	1984, 2007, 2015, 2016, 2017, 2019, 2020, 2021, 2022, 2023	Question Classifier, KGQA Dataset, Research Questions, Other
WH-Patterns	9	5	Scholarly, Open, Covid, Language specific	2000, 2017, 2022, 2023	Question Classifier, KGQA Dataset
Specialized KB Types	2	3	Scholarly, Open	2019, 2021, 2023	KGQA Dataset
Research Focus	8	2	Scholarly, Software Engineering	2003, 2024	KGQA Dataset, Research Questions
Answer Credibility	5	5	Requirements Engineering, Social Science, Healthcare, Open	1984, 2020, 2022, 2023	KGQA Dataset, Question Classifier, Research Questions
Question Goal	7	7	Software Engineering, Design Science, Open	2000, 2003, 2008, 2016, 2019, 2022	Research Questions, Question Classifier
Application Specific	12	5	Design Science, Language specific, Spoken NLP, Open	2000, 2015, 2017, 2019, 2021	Research Questions, Question Classifier, KGQA Dataset

Table 6.4.: Clustering Results for the Extracted Question Types from the Literature

Category 4 This category groups questions based on their interrogative form, distinguishing between different "Wh" word patterns such as *what*, *where*, *which*, *when*, *who*, *why*, *whose*, and *whom*. It is rooted in early question classification research and continues to be relevant in current studies. The cluster is derived from five different sources [104, 132, 127, 133, 105] and spans both scholarly and general domains, with additional contributions from specialized areas such as Covid-related research and studies focused on the Vietnamese language. We refer to this fourth category as *WH-Patterns*.

Category 5 We named this category *Specialized Knowledge Base Types*, as it comprises question classes tailored to specific KBs. These include classes related to ORKG, SPARQL queries, and Wikidata qualifiers. Consequently, such classifications are only applicable when the question is executed against the corresponding KB. The category is supported by three recent sources [104, 129, 31], spanning both scholarly and general domains. Notably, it is exclusively addressed within the context of KGQA dataset literature.

Category 6 We refer to the category as *Research Focus* as it centers on classifying questions according to their research-related focus. It includes classes such as Research Output, Development Methods, and Modeling Approaches and is supported by only two publications. The first is [128], a 2024 contribution in the KGQA dataset category, which specifically targets scholarly content. The second is [125], a 2003 study centered on research questions within the software engineering domain.

Category 7 We refer to this category as *Answer Credibility*, as it classifies questions based on the perceived truthfulness or reliability of the expected answer. It includes classes such as *factual*, *opinion*, *debate*, *conversational*, and *predictive*. This cluster is supported by five sources [37, 130, 131, 106, 136], spanning domains such as requirements engineering, social sciences, healthcare, and the open domain. Most sources are from recent years, with the earliest dating back to 1984. The category is considered in KGQA dataset literature, question classification studies, and research focusing on scholarly questions.

Category 8 We refer to the eighth category as *Question Goal*, as it captures the overarching intent or purpose behind a question. It includes classes such as *exploratory*, *reasoning*, *problem solving*, and *gap spotting*. This category is supported by seven sources [126, 107, 135, 125, 130, 105, 138], spanning domains such as software engineering, design science, and the open domain. The publication years range from 2000 to 2022, reflecting both foundational and more recent work. The category is primarily addressed in question classification studies and literature focused on research question formulation.

Category 9 We refer to this category as *Application Specific*, as it encompasses classes that are tailored to particular applications or use cases. These include categories such as Research Question Utilization, Research Question Typology, and Outcome Artifact Classification. The category is supported by five sources [107, 133, 134, 105, 129], spanning domains such as design science, Vietnamese language studies, spoken natural language processing, and the open domain. The publication years range from 2000 to 2021. This category is addressed

in question classification studies, research question-focused literature, and KGQA dataset research.

6.5. Relevance Assessment of Clusters

In this section, we present the Relevance Assessment phase of the taxonomy construction approach. For each category presented in the prior Section 6.4, we discuss the relevance to the taxonomy. This assessment is based on the established name and description of the category to determine if it aligns with the goal of the taxonomy. In particular, a category is seen as relevant if each of the following questions hold true:

GQ1. Does the category capture characteristics of questions to test the capabilities of KGQA retrieval systems, or does it reflect aspects specific to the scholarly literature search domain?

GQ2. Can the category be generalized across different KGQA systems and within the literature search domain?

GQ3. Can the classes within the category inform and guide the construction of diverse and meaningful evaluation datasets?

In the following, we assess each category towards the questions.

Category 1 – Graph Representation
This category captures structural aspects of question complexity by distinguishing between questions that require the retrieval of single versus multiple facts. This distinction directly impacts the complexity of the retrieval process and is broadly applicable across KGQA systems. Moreover, it is particularly relevant in the scholarly domain, where complex questions are common. The cluster also offers concrete guidance for the construction of datasets that vary in granularity with respect to graph traversal.
Relevant: **Yes**

Category 2 – Answer Type
This category categorizes questions by the expected type of answer, such as dates, entities, quantities, or boolean values. These categories are helpful in describing the characteristics of questions in a literature search to construct diverse datasets. Furthermore, they are broadly applicable across domains and KGQA systems.
Relevant: **Yes**

Category 3 – Question Types
This category focuses on logical and computational operations implied by a question, such as comparison, counting, or negation. These operations are central in retrieval and directly relate to the functional requirements of a retriever. The cluster generalizes across domains and offers valuable guidance for the inclusion of questions targeting different retrieval abilities.
Relevant: **Yes**

Category 4 – WH-Patterns
Although WH-patterns are commonly used in question classification, they primarily reflect linguistic surface forms rather than the underlying retrieval characteristics. Furthermore, the classes are not general enough to provide meaningful characteristics in the scholarly literature search setting. Its utility for guiding the construction of a diverse dataset is also limited. While it may promote diversity with regard to WH-forms, it is not helpful to capture the meaning of questions or reliably correspond to distinct retrieval challenges.
Relevant: **No**

Category 5 – Specialized Knowledge Base Types
This category encompasses classes that are specific to certain knowledge bases, such as Wikidata qualifiers or ORKG-specific constructs. Although these distinctions are meaningful within their contexts, they do not generalize across KGQA systems or domains. Consequently, the cluster offers limited value for constructing broadly applicable evaluation datasets and is more reflective of system-specific constraints than question-inherent properties.
Relevant: **No**

Category 6 – Research-Focus
This category categorizes questions based on their orientation towards research-related topics such as research methods or modeling approaches. Although the classes may be domain-specific, they align closely with our focus on scholarly literature search. In addition, the cluster is relevant for designing evaluation datasets that capture the diversity of scholarly questions.
Relevant: **Yes**

Category 7 – Answer Credibility
This category considers the credibility of expected answers, such as factual, predictive, or opinion-based. Because the literature search can include various credibility types, this classification is relevant to the scholarly literature search and generalizable across different domains. Moreover, considering this classification promotes diversity in the creation of question datasets.
Relevant: **Yes**

Category 8 – Question Goal
This category addresses the underlying intent behind a question, such as problem solving, reasoning, or exploration. These types of goals are relevant for scholarly literature search to capture the intent behind the question. As such, it promotes diversity in question dataset creation.
Relevant: **Yes**

Category 9 – Application-Specific
This category includes classes that are defined in relation to specific applications or use contexts, such as outcome artifact classification or research question typologies. Although these categories may be useful within individual domains, they do not reflect generalizable properties of the questions themselves.
Relevant: **No**

Based on the analysis of category relevance, we conclude that six out of a total of nine identified categories can be considered relevant. Each of these categories further comprises a number of classes. For each individual class, a relevance assessment was also performed, with the detailed results documented in our replication package [124]. In total, 96 classes within the categories were examined, of which 58 were rated as relevant and 38 as not relevant. This yields a class-level relevance ratio of:

$$\text{Relevance}_{\text{classes}} = \frac{|\text{relevant classes}|}{|\text{total classes}|} = \frac{58}{96} \approx 0.6$$

This indicates that approximately 60% of the classes extracted from the literature are considered relevant for our taxonomy. A similar pattern emerges at the category level. The relevance ratio for categories is given by

$$\text{Relevance}_{\text{categories}} = \frac{|\text{relevant categories}|}{|\text{total categories}|} = \frac{6}{9} = \frac{2}{3} \approx 0.66$$

This indicates that a substantial portion of the identified categories are meaningful for structuring our taxonomy.

6.6. Taxonomy Construction and Refinement

This section documents the Taxonomy Construction and Refinement phase of the taxonomy construction approach. The first increment of the taxonomy was initialized based on the relevant categories from Section 6.5. We then identified several shortcomings of the taxonomy, which led to two more refinement steps until we were satisfied with the result. In the following sections, we describe each of the three increments that led to the final KGQA retrieval taxonomy.

6.6.1. First Taxonomy Increment

The following sections will introduce the construction and validation of the first taxonomy increment.

6.6.1.1. Construction of the First Taxonomy Increment

According to the applied construction approach, the first increment of the taxonomy should be created solely using the categories classified as relevant in the Relevance Assessment phase. As such, the first increment of the taxonomy $\mathcal{T}_1$ is initialized with the categories that we consider relevant based on the consideration of relevance in Section 6.5. The first increment of the taxonomy is illustrated in Table 6.5 and consists of six categories and 58 classes.

Category	Classes		
Graph Representation	Single Fact		Multi Fact
Answer Type	Undefined	Date	Tool/Notation
	Definition	Theoretical Framework	
	Abbreviation	Instructional	Actor
	Boolean	Entity	Human/Person
	Quantitative	Solution	Time
	Title	Monetary	Name
	Properties	Description	Organization
	Technology	Software System	Duration
	Location	Distance Measurement	
	Other	Bibliometric Numbers	
	Procedure/Technique		
Question Type	Negation	Dependency	Contingency
	Superlative	Comparison	Listing
	Ranking	Multiple Intentions	Temporal
	Relationship	Counting	Aggregation
Research Focus	Development Methods	Qualitative Modeling	Analytical Methods
	Generalization	Analytic Modeling	Empirical Modeling
Question Goal	Exploratory	Reasoning	Gap Spotting
	Problematization	Method Improvement	
	Problem Solving		
Answer Credibility	Factual	Debate	Conversational
	Predictive	Opinion	

Table 6.5.: The categories and classes of the first taxonomy increment $\mathcal{T}_1$

6.6.1.2. Validation of the First Taxonomy Increment

The validation is guided by the questions **Q1-5** which are presented in the GQM plan shown in Table 5.1. We indicate the taxonomy as $\mathcal{T}_1$, the classes of the taxonomy as $C_{\mathcal{T}_1}$, and the objects under study as C, which are the classes that were extracted from the literature survey. The generality (**Q1.1**) concerns whether the taxonomy is general enough, measured using *laconicity* (**M1.1.1**) and *lucidity* (**M1.1.2**). The laconicity measures $laconicity(\mathcal{T}_1, C) = \frac{227}{227} = 1.00$, which is the maximum score that can be reached. Therefore, this indicates that no two classes from the literature appear more than once in the taxonomy, suggesting that there is no redundancy. This is to be expected given the cluster-based approach that was applied to initialize the taxonomy because each cluster was derived directly from classes extracted in the literature and no literature class is included in more than one cluster. Only the presence of such an overlap would reduce the laconicity score. Regarding the lucidity with the given $C_{\mathcal{T}_1}$ and C, we measure $lucidity(C_{\mathcal{T}_1}, C) = \frac{28}{58} = 0.48$. Based on this result, we can see that approximately half of the classes in the taxonomy are referenced by more than one literature source. Even though the interpretation of the metric

Question	Metric	Result
Q1.1 Generality	**M1.1.1** Laconicity	1.00
	M1.1.2 Lucidity	0.48
Q1.2 Appropriateness	**M1.2.1** Completeness	0.59
	M1.2.2 Soundness	1.00
Q1.3 Orthogonality	**M1.3.1** Orthogonality Matrix	0.96
Q2.1 Relevance	**M2.1.1** Fraction of Relevance	1.00
Q2.2 Novelty	**M2.2.1** Innovation	0.00
	M2.2.2 Adaption	1.00

Table 6.6.: Results of the validation for $\mathcal{T}_1$

would indicate a bad generality, we argue that in our case, this is a positive indication as it demonstrates that a significant portion of the taxonomy is supported by multiple sources.

The next question **Q1.2** is about the appropriateness of the taxonomy. We measure the *completeness* (**M1.2.1**) score as $completeness(C_{\mathcal{T}_1}, C) = \frac{135}{227} = 0.59$. The score indicates that more than half of the types that we extracted from the literature have been included in some way in the taxonomy. The remaining types are not relevant to our taxonomy and, as such, are excluded from the taxonomy. For the *soundness* (**M1.2.2**), we measure $soundness(C_{\mathcal{T}_1}, C) = \frac{58}{58} = 1.00$. The score is maximal, which is to be expected, as only in cases where a class from the taxonomy would not map to a class from the literature the soundness metric would be negatively affected and this does not happen because of the clustering approach.

Question **Q1.3** addresses the concept of *orthogonality*, which quantifies the number of overlapping classes within the taxonomy. We assess *orthogonality* (**M1.3.1**) using an orthogonality matrix, calculated as $orthogonality(C_{\mathcal{T}_1}, C) = \frac{3176}{3306} = 0.96$, where the numerator represents orthogonal classes without overlap. Out of 3306 cases, 130 instances are identified as overlapping. We identified a total of 10 overlaps between the *Question Type* category and the *Multi Fact* class. These overlaps arise because certain operations, such as *Dependency*, *Listing*, or *Aggregation*, require multiple facts from the graph. We acknowledge these overlaps but consider them acceptable, as the categories capture distinct but relevant dimensions of question classification in the context of KGQA. Within the category *Question Type* itself, additional overlaps were observed. Specifically, both *Contingency* and *Dependency* imply the presence of *Relationship*, making this classification redundant. Therefore, a refinement for these classes is needed. There is also overlap between the classes *Superlative*, *Ranking*, and *Counting*. All of these classes are based on the counting operation. However, they differ in terms of complexity and retrieval intent. Therefore, while they share underlying operations, each class represents a distinct capability in the retrieval process. Therefore, we argue in favor of maintaining their separation despite functional similarities.

Moreover, we identify a high degree of overlap involving the classes *Entity* and *Name*. These classes are intended to classify questions that inquire about a specific entity. Given that we assume that an entity always has a name, the expected answer to the question

involves naming this entity, which is why the number of overlaps is high. These overlaps include 13 classes from the *Answer Type* category, including *Technology*, *Human/Person*, *Software System*, or *Location*. Additionally, all classes in the *Research Focus* category overlap with *Entity* and *Name*. These observations highlight the need to refine both categories to reduce redundancy. Overlaps are also found in the *Answer Credibility* category. First, *Debate* is about the provocation of discussion or argumentation rather than providing a factual answer. This naturally aligns it with the *Opinion* class. Second, the *Predictive* class can be based on facts or opinions, which makes the class redundant to the *Factual* and *Opinion* classes. In addition, predictions can represent the goal of a question rather than a type of answer credibility. For these reasons, we suggest relocating *Predictive* to the *Question Goal* category. In the category *Question Goal*, we identify redundancies for the classes *Gap Spotting* and *Problematization*, as both seek to identify shortcomings or problems in the existing literature. Therefore, we see the need to combine these classes. Furthermore, we identify 36 overlaps with the *Exploratory* class, which can be attributed to its broad definition. Given the nature of many KGQA questions, one could argue that their primary purpose is to explore the KG in order to retrieve answers. We therefore argue that the *Exploratory* class should be removed due to its overly general nature.

The next question **Q1.4** is about the *relevance* (**M2.1.1**) of the taxonomy. Because we initialized the taxonomy based on a relevance analysis, we argue that all classes in the taxonomy are relevant to our cause. Therefore, we measure $relevance(C_{\mathcal{T}_1}, C) = \frac{58}{58} = 1.00$, which is the maximal score.

Whether the taxonomy reached an appropriate novelty (**Q1.5**) is measured with the innovation and adaptation metrics. Concerning *innovation* (**M2.2.1**), we measure $innovation(C_{\mathcal{T}_1}, C) = \frac{0}{58} = 0.00$, the lowest possible value. With regard to *adaption* (**M2.2.2**), we measure $adaptation(C_{\mathcal{T}_1}, C) = \frac{58}{58} = 1.00$, the highest possible value. This means that the taxonomy does not provide innovation and is fully adapted from the literature. This is an expected outcome, as the first increment of the taxonomy consists only of categories and classes from the literature.

6.6.2. Second Taxonomy Increment

When validating the first taxonomy increment $\mathcal{T}_1$, we found that there are issues with orthogonality. Some of the classes have overlaps, leading to redundancies and difficulties in the application. To solve these problems, we create the second increment $\mathcal{T}_2$ of the taxonomy, which is shown in Table 6.7. In the following, we discuss the changes and validate the result.

6.6.2.1. Construction of the Second Taxonomy Increment

The construction of the second taxonomy increment $\mathcal{T}_2$ is based on a refinement of the issues identified in the first increment $\mathcal{T}_1$. In the following, we discuss the changes that were applied.

Category	Classes		
Graph Representation	Single Fact		Multi Fact
Answer Type	Named Entity	Description	Temporal
	Quantitative	Boolean	Other Type
Answer Format	Simple	Enumerative	Other Format
	Explanatory		
Question Type	Negation	Counting	Aggregation
	Superlative	Comparison	Relationship
	Multiple Intentions	Ranking	Temporal
Question Goal	Problem Solving	Reasoning	Improvement
	Prediction	Problematization	
Answer Credibility	Subjective	Objective	

Table 6.7.: The categories and classes of the second taxonomy increment $\mathcal{T}_2$

Consolidation of Entity-Related Classes In the first increment of the taxonomy ($\mathcal{T}_1$), it became evident that the classes *Entity* and *Name* exhibit a significant overlap within the taxonomy. This can be attributed to the fact that both classes are defined in a highly generic manner, and many more specific classes ultimately involve identifying the name of an entity. For example, questions concerning a software tool, an organization, or a scientific paper can all be interpreted as inquiries about nameable entities.

To address this in the second increment ($\mathcal{T}_2$), we consolidated these overlapping classes under a new, higher-level class: *Named Entity*. This decision was guided by the goal of the *Answer Type* category, which is to classify questions based on the type of information expected in the answer. Questions such as "Which software is used for X?" or "What is the title of the paper from Y?" illustrate different entities but share the commonality of seeking a name. Furthermore, highly granular classes like *Software Tool*, *Organization*, or *Paper* reduce the generalizability of the taxonomy, as this abstraction level makes it difficult to reach a state of completeness. Moreover, such fine distinctions do not meaningfully impact the retrieval process, as the central objective remains the identification of the name of an entity. Consequently, we argue that the specific semantic content of the name is irrelevant to the underlying retrieval mechanics.

The consolidation into the *Named Entity* class encompasses a total of 18 classes. This includes all classes from the *Research Focus* category and 11 classes from *Answer Type*. We also merged the *Date*, *Time*, and *Duration* classes into a single class *Temporal*, as all of them classify by a specific point or range in time. Moreover, we merged all quantification classes into the *Quantitative* class to reduce redundancy and keep the same level of abstraction across the category. Lastly, we consolidated the classes *Theoretical Framework*, *Tool/Notation*, *Description*, and *Definition* into a single class named *Description*. This new type should capture those answers that are descriptive in nature, meaning that their goal is to describe or explain something using one or more comprehensive natural language sentences.

The classes in the *Answer Type* category are now *Named Entity*, *Description*, *Temporal*, *Quantitative*, *Boolean*, and *Other Type*. We consider these classes to be orthogonal to each other and argue that each represents different complexities in the KGQA retrieval process. This allows for the classification of questions according to whether they seek a name, a description, a date or time span, quantitative data, Boolean data, or another type of information..

Introduction of the Answer Format Category The second increment of the taxonomy ($\mathcal{T}_2$) introduces a new category, which we refer to as *Answer Format*. This category classifies questions according to their expected format or structure of the answer. It emerged from the observation that three classes in $\mathcal{T}_1$ described the format of the answer rather than its content, making their original categorization semantically inconsistent. This applies to the class *Instructional*, which classifies whether an answer is expected to be a step-by-step sequence of instructions. Furthermore, it concerns the classes *Properties* and *Listing*, both of which expect an answer in the form of an enumeration. These three are therefore combined into a new class called *Enumerative*.

To increase the completeness of the new category, we introduce three additional classes: *Simple*, *Explanatory*, and *Other Format*. The class *Simple* classifies questions that expect a direct, brief, and minimalistic answer. These are typically answers consisting of a single sentence that conveys a factual statement without further explanation. The class *Explanatory* classifies questions for which the expected answer is a textual explanation of a phenomenon or an entity. Finally, we introduce the new class *Other Format* to cover all answer formats that have not been considered in the category thus far.

Refinement of Answer Credibility In the second increment $\mathcal{T}_2$, we refine the *Answer Credibility* dimension by reducing it to the classes *Subjective* and *Objective*. The classes *Debate* and *Opinion* were merged into *Subjective*, while *Factual* was renamed to *Objective* to clearly contrast with subjective responses. This change aims to improve clarity by emphasizing the binary nature of the perceived credibility of the answer.

Adjustments to Question Goal During the validation of the first taxonomy increment ($\mathcal{T}_1$), the *Exploratory* class was found to be overly broad, leading to significant overlaps with other classes. As most research questions are exploratory to some degree, we argue that the class lacks discriminative power, which is why we removed it.

Furthermore, the classes *Problematization* and *Gap Spotting* were merged, as both aim to identify shortcomings or gaps in existing literature. Moreover, to resolve inconsistencies in the abstraction levels in the category, we renamed *Method Improvement* to *Improvement*, aligning it with the general abstraction level used in the category.

Lastly, the *Prediction* class, originally part of *Answer Credibility*, was moved to *Question Goal*. This shift reflects its stronger alignment with the intent of the question rather than the credibility of the answer and helps to complete the conceptual space covered by *Question Goal*.

Question	Metric	Result
Q1.1 Generality	**M1.1.1** Laconicity	1.00
	M1.1.2 Lucidity	0.25
Q1.2 Appropriateness	**M1.2.1** Completeness	0.57
	M1.2.2 Soundness	0.89
Q1.3 Orthogonality	**M1.3.1** Orthogonality Matrix	0.98
Q2.1 Relevance	**M2.1.1** Fraction of Relevance	1.00
Q2.2 Novelty	**M2.2.1** Innovation	0.11
	M2.2.2 Adaption	0.89

Table 6.8.: Results of the validation for $\mathcal{T}_2$

6.6.2.2. Validation of the Second Taxonomy Increment

Guided by the questions **Q1-5** from the GQM plan shown in Table 5.1, we are discussing the validation of the second increment of the taxonomy ($\mathcal{T}_2$). Regarding the *lucidity* (**M1.1.2**), the new increment measures $lucidity(C_{\mathcal{T}_2}, C) = \frac{7}{28} = 0.25$, which is 0.23 less than in the first increment. This drop in score is the result of the merges of classes, as 21 out of 28 classes in the taxonomy can be mapped to more than one class from the literature. Still, we consider these merges to be justified because we were able to reduce a large number of redundancies, making the taxonomy more interpretable and usable. Moreover, as we argued before, we consider a low *lucidity* to be good in our case, as this indicates that a high number of classes in our taxonomy are supported by multiple sources in the literature. The *laconicity* (**M1.1.1**), on the other hand, stays the same as in the first increment $\mathcal{T}_1$.

We argue that regarding the *generality* (**Q1.1**) of the taxonomy, we reached an appropriate level. By applying the merges to the classes, we were also able to reduce the number of classes from 58 to 28 classes, an overall reduction of approximately 52%. We assert that the remaining classes provide a high level of generality, making it less likely that classes are present that are never used during application.

With regard to the *appropriateness* (**Q1.2**), we measure *completeness* (**M1.2.1**) $completeness(C_{\mathcal{T}_2}, C) = \frac{130}{227} = 0.57$. This is a reduction of 0.02 compared to the first increment $\mathcal{T}_1$, which is the result of the removal of the *Exploratory* class, as this class corresponds to five classes from the literature. However, we see this removal as justified, as it reduces the overall amount of dependencies, which improves the orthogonality of the taxonomy.

Furthermore, for the *soundness* (**M1.2.2**) we measure $soundness(C_{\mathcal{T}_2}, C) = \frac{25}{28} = 0.89$, which is a reduction of 0.11 in comparison to the first increment $\mathcal{T}_1$. This is the result of introducing three new classes: *Simple*, *Explanatory* and *Other Format* to the taxonomy, which are not covered by the literature. We argue that the additions improve the overall generality of the taxonomy, making it more reliably applicable.

We measure *orthogonality* (**M1.3.1**) using the orthogonality matrix as $orthogonality(C_{\mathcal{T}_2}, C) = \frac{745}{756} = 0.98$. We observe that 11 classes have overlaps,

which is a significant reduction from the previous 130 overlaps identified in the first increment $\mathcal{T}_1$. The remaining overlaps concern classes from the *Question Type* category with the *Multi Fact* class and within the *Question Type* category itself. As we argued before, we think that this overlap is justified because of the differences in complexity that they impose on the retrieval.

Regarding *relevance* (**M2.1.1**), we argue that the relevance of previous classes remains the same. Furthermore, we identify the three new classes that have been added in this increment as relevant for KGQA retrieval, as they assess the ability of the retriever to present the answers in different ways. Therefore, we measure the $relevance(C_{\mathcal{T}_2}, C) = \frac{28}{28} = 1.00$.

The addition of three new classes increased the *innovation* (**M2.2.1**) of the taxonomy which we measure as $innovation(C_{\mathcal{T}_2}, C) = \frac{3}{28} = 0.11$. Consequently, it reduced *adaptation* (**M2.2.2**) by 0.11, which we measure as $adaptation(C_{\mathcal{T}_2}, C) = \frac{25}{28} = 0.89$.

In summary, scores with respect to *generality*, *appropriateness* and *orthogonality* reached a satisfactory point. However, regarding the novelty of the taxonomy, we observe that there is a need to add some missing classes to improve the applicability of the taxonomy. Furthermore, we identify that the *Question Type* dimension contains multiple intentions and may be subject to a name change. As such, we conduct a third increment to improve these areas of the taxonomy.

6.6.3. Third Taxonomy Increment

In the third increment of the taxonomy, two new categories are introduced: *Condition Constraint* and *Intention Count*. In addition, four new classes are added: *Basic*, *Other Goal*, *Information Lookup Normative* and *Single Intention*. Furthermore, the *Question Type* category has been renamed to *Retrieval Operation*. We discuss these changes in the following.

6.6.3.1. Construction of the Third Taxonomy Increment

The construction of the third taxonomy increment $\mathcal{T}_3$ is based on a refinement of the issues identified in the second increment $\mathcal{T}_2$. In the following, we discuss the changes that were applied.

Introduction of the Condition Type Category We have identified a gap in the existing classification scheme: currently there is no category that captures the types of conditions that an answer must satisfy. These constraints are important because they can significantly affect the complexity of answering a question. In particular, they offer insight into how well a KGQA approach handles multiple or diverse types of conditions.

To address this, we introduce a new dimension called *Condition Type*, which classifies questions based on the types of conditions or constraints that the expected answer must meet. We hypothesize that these condition types mirror those found in the *Answer Type* dimension, since both describe similar categories of information. However, we make an

Category	Classes		
Graph Representation	Single Fact		Multi Fact
Answer Type	Named Entity	Description	Temporal
	Quantitative	Boolean	Other Type
Condition Type	Named Entity	Description	Temporal
	Quantitative	Other Type	
Answer Format	Simple	Enumerative	Other Format
	Explanatory		
Retrieval Operation	Basic	Counting	Aggregation
	Superlative	Comparison	Relationship
	Negation	Ranking	
Intention Count	Single Intention	Multiple Intentions	
Question Goal	Problem Solving	Reasoning	Improvement
	Prediction	Problematization	Information Lookup
	Other Goal		
Answer Credibility	Subjective	Objective	Normative

Table 6.9.: The categories and classes of the third taxonomy increment $\mathcal{T}_3$

exception for the *Boolean* class, as we argue that a condition by nature is Boolean, so this class does not fit into the new category. Therefore, we initialize the new category *Condition Type* with the same set of classes as *Answer Type* except for *Boolean*.

Renaming of the Question Type Category We observed that the previous *Question Type* category consists mostly of operations. We have therefore decided to rename the category to *Retrieval Operation*. With this renaming, we want to make it clear to the reader directly from the name what this category is trying to achieve. We believe that the previous name was too abstract to achieve this goal. Furthermore, we argue that the *Temporal* class in the category is not an operation but a constraint. We have therefore merged the class into the *Temporal* class of the condition type category.

Introduction of the Intention Count Category We determined that the *Multiple Intentions* class does not perform an operational function. Rather, it reflects the structural complexity of a question. Since it no longer aligns with the *Retrieval Operation* category, we introduce the *Intention Count* category. This new category classifies questions according to the number of distinct intentions they express. If a question contains a single, unified query that cannot be decomposed into separate independent questions without altering its context, it is classified as *Single Intention*. In contrast, if a question can be split into two or more independent queries, it falls under *Multiple Intentions*.

Adding Missing Classes We identified four missing classes in our taxonomy and introduced them in this increment. First, *Basic* was added to the *Retrieval Operation* category to capture questions in which the retriever should simply extract one or more factual items from the

Question	Metric	Result
Q1.1 Generality	**M1.1.1** Laconicity	0.70
	M1.1.2 Lucidity	0.32
Q1.2 Appropriateness	**M1.2.1** Completeness	0.57
	M1.2.2 Soundness	0.78
Q1.3 Orthogonality	**M1.3.1** Orthogonality Matrix	0.99
Q2.1 Relevance	**M2.1.1** Fraction of Relevance	1.00
Q2.2 Novelty	**M2.2.1** Innovation	0.22
	M2.2.2 Adaption	0.78

Table 6.10.: Results of the validation for $\mathcal{T}_3$

graph, without requiring further processing. Second, we introduced *Normative* into the category *Answer Credibility* to classify questions based on ethical, normative, or political values. Third, we added *Information Lookup* to classify questions whose objective is simply to retrieve specific data from the KG into the *Question Goal* category. Finally, *Other Goal* was included to accommodate questions that do not fit any other defined goal class, ensuring that all questions can be properly classified.

6.6.3.2. Validation of the Third Taxonomy Increment

With regard to *laconicity* (**M1.1.1**), we measure the following $laconicity(C_{\mathcal{T}_3}, C) = \frac{160}{227} = 0.70$, which represents a decrease of 0.30 compared to $\mathcal{T}_2$. The reason for this reduction lies in the fact that, for the new category *Condition Type*, we use the same classes as in the category *Answer Type*. As a result, the metric detects a high degree of redundancy between our taxonomy and the classes from the literature. However, we consider the separation of these two categories to be justified since they address different aspects that are both essential for representing the complexity of a KGQA retrieval.

For *lucidity* (**M1.1.2**), we measure $lucidity(C_{\mathcal{T}_3}, C) = \frac{12}{37} = 0.32$. This corresponds to an increase of 0.07 compared to the second increment $\mathcal{T}_2$. The improvement is due to the addition of new classes that are not found in the literature, which positively influences the score. In general, we argue that the low *lucidity* value is actually a positive indication for the taxonomy. This is because a high overlap with the literature indicates that our taxonomy aligns well with existing research.

In general, we argue that the *generality* (**Q.1.1**) has reached an appropriate level. By consolidating the classes, we achieved a consistent level of abstraction within the categories. This enables the taxonomy to be applied in a general way while maintaining enough precision to support the classification of both the capabilities of a KGQA retrieval system and the characteristics relevant for scholarly literature search.

The *completeness* (**M1.2.1**) is measured as $completeness(C_{\mathcal{T}_3}, C) = \frac{130}{227} = 0.57$ and remains unchanged compared to the previous increment $\mathcal{T}_2$. In contrast, *soundness* (**M1.2.2**) shows a

value of $soundness(C_{\mathcal{T}_3}, C) = \frac{29}{37} = 0.78$, which represents a decrease of 0.11 compared to the previous increment. This change can be attributed to the addition of new classes, as these are not mapped to classes from the literature.

In general, we argue that the *appropriateness* (**Q1.2**) has reached a satisfactory level. From the set of classes found in the literature, we have filtered out those that are relevant to our taxonomy. A higher level of *completeness* could be achieved by including more classes from the literature in the taxonomy. However, this would negatively affect other metrics since those additional classes are not relevant to our cause. Furthermore, *soundness* correlates with the *novelty* of the taxonomy. This means that removing new classes from the taxonomy would increase the value but at the cost of reducing the applicability and usefulness of the taxonomy.

The amount of overlaps (**Q1.3**) in the taxonomy is measured with the *orthogonality* (**M1.3.1**), which we measure as $orthogonality(C_{\mathcal{T}_3}, C) = \frac{1321}{1332} = 0.99$, amounting to 11 overlaps across the taxonomy. These overlaps are the same as in $\mathcal{T}_2$, where we already argued their appropriateness.

As with previous increments, the *relevance* (**M2.1.1**) measures $relevance(C_{\mathcal{T}_3}, C) = \frac{37}{37} = 1.00$. This is because the previous classes in the taxonomy were already determined to be relevant and the new additions were included because they were missing and are relevant to our objectives.

Regarding the *novelty* (**Q2.2**) of the taxonomy. The *innovation* (**M2.2.1**) measures $innovation(C_{\mathcal{T}_3}, C) = \frac{8}{37} = 0.22$, indicating that 8 of the 37 classes are new, not yet considered in the classes from our literature survey. Furthermore, the *adaptation* (**M2.2.2**) measures $adaptation(C_{\mathcal{T}_3}, C) = \frac{29}{37} = 0.78$, which states that 29 of the 37 classes were adapted from the literature.

Overall, we are satisfied with the *novelty* of the third increment $\mathcal{T}_3$ of the taxonomy. We could try to find more new classes to reach a higher *innovation* score, but this would also introduce the risk of lowering the orthogonality and generality of the taxonomy.

6.7. Categories and Classes of the Taxonomy

In Section 6.6, three different increments of the taxonomy were created in an iterative process. With the last increment being $\mathcal{T}_3$, which is presented in Table 6.9, we have reached a satisfactory point by validating the generality (**Q1.1**), appropriateness (**Q1.2**), orthogonality (**Q1.3**), relevance (**Q2.1**), and novelty (**Q2.2**). We therefore introduce the categories and classes of the final KGQA retrieval taxonomy in this section, before demonstrating its application in the following Section 6.8.

Class	Description
Single Fact	Whether the answer can be found in a single fact of the KG.
Multi Fact	Whether the answer must be aggregated from multiple facts in the KG.

Table 6.11.: Graph representation category of the KGQA retrieval taxonomy. It classifies questions by the amount of facts that need to be retrieved to provide a complete answer.

6.7.1. Graph Representation

The goal of the **Graph Representation** category is to classify the number of facts required to answer a given question. Here, a fact in a graph is formed by two entities connected through a relationship [140]. Specifically, in a RDF graph, such a fact is represented as a triple consisting of $(subject, predicate, object)$ [35].

The category comprises two distinct classes, as shown in Table 6.11. The *Single Fact* class is assigned when only one fact is needed, whereas the *Multi Fact* class is used if multiple facts must be combined to construct the answer. These two classes are mutually exclusive, meaning that a question can be assigned to one of them only. Furthermore, it is important to note that in order to apply this category appropriately, it is essential to have knowledge of the underlying structure of the KG. Specifically, it must be determined whether the answer to the question is represented by a single triple or spans across multiple triples within the graph.

6.7.2. Answer Type

The **Answer Type** category serves to systematically classify questions based on the types expected in their answers. Consequently, applying this category necessitates the formulation of hypothetical answers, since the classification is determined by the inherent types present in the anticipated response. Furthermore, multiple classifications are allowed and may even be necessary for complex questions. A total of six distinct classes are defined, as presented in Table 6.12.

Named Entity is applied when the answer contains specific named entities. This includes persons, organizations, geographical locations, as well as formal identifiers such as publication titles or Digital Object Identifiers (DOIs). An example of such a question is “What is the long form of MVC?”, which in the context of software engineering stands for “Model-View-Controller”. Since this is the name of a popular software architectural style, we can classify that the expected answer to this question is a named entity.

Description classifies expected answers that consist of explanatory or definitional natural language expressions. These answers are formulated in complete sentences and provide

Class	Description
Named Entity	The expected answer contains a specific named entity, such as people, organizations, locations, systems, technologies, and other concrete subjects or objects that are usually capitalized.
Description	The expected answer contains a descriptive phrase or sentence, such as explanations, definitions, theoretical constructs, or conceptual overviews.
Temporal	The expected answer refers to a specific point in time or a time range, such as dates, timestamps, or general time expressions.
Quantitative	The expected answer contains a numerical value, such as a measurements, monetary values, or general counts.
Boolean	The expected answer is binary, typically "yes" or "no".
Other Type	The expected answer is of a type that does not fit into any of the other defined classes.

Table 6.12.: Answer type category of the KGQA retrieval taxonomy. It classifies questions by the types that are contained within the expected answer.

detailed information about concepts, processes, or relationships. An example of this classification is the question "What does the MVC architecture pattern do?". Because this question expects the answer to be a complete sentence explanation in natural language, it qualifies for this class.

Temporal refers to expected answers that involve temporal expressions, either as specific points in time or as intervals of time. For example, the question "When was the MVC software architecture pattern first mentioned?" asks for a specific publication date. Because the answer is expected to provide temporal information, the class qualifies.

Quantitative includes expected answers that contain numerical information. This can include absolute numbers, units of measurement, statistical values, or comparative quantities. An example of a question that qualifies for this class is "How much does the application of the MVC software architectural style affect the response time of the system?". The answer to this question is expected to be a number that quantifies the change in response time. Since this is a quantitative type, the class qualifies.

Boolean denotes expected answers that represent a binary decision. Typically, these are statements that can be answered with "yes" or "no". This class is particularly useful for decision-making questions such as "Does using the MVC software architecture pattern

Class	Description
Named Entity	The question contains a condition that requires the answer to involve a specific named entity, such as people, organizations, locations, systems, technologies, and other concrete subjects or objects that are usually capitalized.
Description	The question contains a description of a condition that the answer must meet, such as explanations, definitions, theoretical constructs, or conceptual overviews.
Temporal	The question contains a condition that requires the answer to be limited to a specific time or time range, such as dates, timestamps, or general time expressions.
Quantitative	The question contains a condition that requires the answer to be limited to a specific numeric expression, such as a measurement, monetary value, or general count.
Other Type	The question contains a condition that does not fit into any of the other defined classes.

Table 6.13.: The condition type category of the KGQA retrieval taxonomy. It classifies questions by the types of conditions contained within them.

improve the maintainability of the system?". Since the answer is expected to be a binary yes or no, the class qualifies.

Other Type is used for answer types that do not clearly fit into any of the aforementioned classes. It serves as a fallback classification.

6.7.3. Condition Type

The **Condition Type** category classifies questions based on the conditions that a valid answer must fulfill. Although it shares five of the six classes with the category **Answer Type**, as shown in Table 6.13, its focus lies on the constraints and requirements expressed within the question rather than the types of the expected answer itself. Because a question can impose multiple different types of constraints, these classes are not mutually exclusive.

Named Entity applies when the question specifies a named entity to which the answer must refer. These typically include persons, organizations, technologies, or other clearly identifiable entities. For example, in the question "What defines a microservice architecture?",

Class	Description
Simple	The expected answer is brief, straightforward, and minimalistic, often consisting of a one-sentence response, a yes/no answer, or a direct factual statement without additional elaboration.
Enumerative	The expected answer is formatted as a list, for example listing the properties or characteristics inherent to a phenomenon, object, or entity.
Explanatory	The expected answer provides a textual explanation about a certain phenomenon, object, or entity.
Other Format	The expected answer format does not fit into any of the other classes.

Table 6.14.: The answer format category of the KGQA retrieval taxonomy. It classifies questions by the expected formatting of the answer.

the term "microservice architecture" constitutes a concrete entity and therefore serves as a conditional constraint. This is because the answer must explicitly name this entity.

Description is used when the question includes a descriptive condition involving conceptual, functional, or relational aspects. An illustrative case is: "What is a software architecture pattern that allows for the development of scalable, modular, and maintainable systems?", which specifies the qualitative requirements that the answer must address.

Temporal refers to conditions that impose temporal constraints, such as the mention of a specific point or range in time. For example, the question "Which software architecture patterns were introduced in 2014?" requires that the answer only includes patterns introduced within that year.

Quantitative is relevant when a question includes numerical constraints. These may be percentages, counts, measurements, or statistical values. For instance, the question "Which software architecture pattern has the potential to improve the scalability of software systems by 20%?" demands an answer that specifically corresponds to this numerical condition.

Other Type serves as a fallback class for conditions that do not clearly fall into any of the aforementioned classes.

6.7.4. Answer Format

The **Answer Format** category classifies questions based on the expected structure and presentation of the corresponding answer. Consequently, applying this category requires

the formulation of a hypothetical answer, as the classification is grounded in the anticipated formatting of the response. In most cases, the assignment to one class will be mutually exclusive. However, for more complex questions that pursue multiple intentions or contain embedded instructional components, assigning multiple classes may be appropriate.

Simple applies to questions for which a brief, direct, and unambiguous response is expected. This often involves a single sentence or even a single word, typically conveying a factual statement without elaboration. For instance, the question "What is the typical first layer in a layered architecture?" qualifies for this class, as it expects a concise, definitive answer.

Enumerative is used when the answer is expected to consist of a list, such as a series of properties, components, or characteristics of a concept or entity. An illustrative example is "What are the typical layers in a layered architecture?" where the anticipated answer involves listing distinct architectural layers.

Explanatory applies to questions that require a multi-sentence textual explanation. These responses provide more elaborate contextualization or definitions and go beyond the conciseness of the *Simple* class. An example is "What is the definition of a layered architecture?", which demands a comprehensive and structured explanation.

Other Format serves as a fallback class for answers that do not clearly conform to any of the previously defined formats.

6.7.5. Retrieval Operation

The **Retrieval Operation** category classifies the questions according to the operations that must be applied to the information contained within the KG in order to derive an answer. In most cases, the assignment of a single class is sufficient. However, for more complex questions that pursue multiple intentions, it may be necessary to assign multiple classes. Furthermore, it is important to note that some classes inherently include operations required by others. In such cases, only the class representing the more complex operation should be assigned. In addition, an accurate classification in this category is only possible when the structure of the underlying KG is known. This is due to the level of granularity at which the data may be stored. For example, some information is contained within a single triple, while other information is distributed across multiple triples. Furthermore, it must be considered whether certain operations are already precomputed and stored in the graph or whether the operation must be performed during retrieval.

Basic refers to questions that require only locating and retrieving a specific piece of information. The answer is directly available in the graph without the need for additional operations. For instance, the question "What is the definition of the Client-Server software architecture pattern?" qualifies as Basic, assuming the answer is stored in a single triple.

Relationship involves identifying connections or dependencies between multiple entities in the graph. This includes causal, hierarchical, or associative relationships that are not stored in a single triple. For example, "Which components in a client-server software architecture need to communicate with each other?" requires uncovering a communication relationship

Class	Description
Basic	The answer can be retrieved directly without applying additional operations.
Relationship	Requires identifying a connection or dependency between pieces of information, such as causalities or correlations.
Negation	Requires identifying where a condition does not hold, based on explicit negation or missing information.
Aggregation	Requires aggregating multiple pieces of information into a single answer.
Counting	Requires counting the number of relevant instances in the data.
Superlative	Requires identifying the most or least of certain information
Ranking	Requires ordering information based on a specific criterion.
Comparison	Requires comparing two or more pieces of information based on common attributes.

Table 6.15.: The retrieval operation category of the KGQA retrieval taxonomy. It classifies questions by the operation that needs to be performed on the data in the KG to arrive at the answer.

between components, which needs to be found by understanding the relationships between the triples.

Negation applies when the answer requires determining the absence of a condition. This can be based on explicit negation or on the absence of expected triples. For example, "Is the system not based on a software architecture pattern?" requires verifying that no such relationship exists in the graph.

Aggregation encompasses questions that involve combining multiple pieces of information into a single response. This includes both qualitative and quantitative aggregations, such as listing characteristics: "What are the characteristics of a client-server architecture?" or computing averages: "What is the average runtime of systems based on the client-server architecture?". The classification assumes that the aggregation is not precomputed in the graph.

Counting applies to questions that require determining the number of instances that satisfy a particular condition. For example, "How many implemented systems are based on the client-server architecture?". This also assumes that the count is not already stored in the graph.

Class	Description
Single Intention	Questions that focus on one objective or query, which cannot be meaningfully divided into separate, independent questions without losing their original context.
Multiple Intentions	Questions that embed multiple distinct objectives or queries, which could be broken up into multiple separate components or questions, each addressing a different aspect.

Table 6.16.: The intention count category of the KGQA retrieval taxonomy. It classifies questions based on the number of distinct intentions embedded within them.

Superlative refers to questions that seek to identify the most or least of something, either in qualitative or quantitative terms. For example, "What is the software architecture design pattern that has been applied the most?" requires calculating frequency or degree of importance. Although this may involve a counting operation, only the *Superlative* class should be assigned in such cases.

Ranking applies to questions that require sorting a set of entities according to a given criterion, which may be quantitative or qualitative. An example is: "What are the systems that are based on the client-server architecture sorted by their runtimes?". The sorting must be performed at retrieval time.

Comparison involves evaluating two or more pieces of information relative to one another based on a shared attribute or criterion. This can be qualitative or quantitative. For instance, "What is the runtime of system X compared to the runtime of system Y?" requires retrieving and comparing individual data points, assuming the comparative conclusion is not already stored.

6.7.6. Intention Count

The *Intention Count* category distinguishes between two mutually exclusive classes, as presented in Table 6.16. A practical criterion for classification is whether the question can be meaningfully split into multiple subquestions without altering its overall intent or expected answer. The following classes are contained within the category:

Single Intention applies to questions that pursue a single, coherent objective. Decomposing such questions would change the meaning or completeness of the original inquiry. An example is: "What even is a software architecture pattern?" The question expresses a singular intention, and splitting it would diminish its clarity or alter the intended response.

Multiple Intentions encompasses questions that consist of multiple distinct objectives or components. Each part could stand alone as an independent question, and collectively, they

Class	Description
Subjective	Questions that expect answers that are not strictly bound by objective evidence.
Objective	Questions that expect answers grounded in verified data, facts, or empirical evidence.
Normative	Questions expecting answers based on norms, ethics, or policies.

Table 6.17.: The answer credibility category of the KGQA retrieval taxonomy. It classifies questions based on the truthfulness of the information in the answer.

retain the original meaning. For example, "What is a software architecture pattern and how can I apply it?" This question seeks both a conceptual explanation and a practical application, representing two separate intentions that can be addressed individually.

6.7.7. Answer Credibility

The **Answer Credibility** category classifies questions based on the nature of truthfulness expected in the corresponding answer, as summarized in Table 6.17. The classes in this category are mutually exclusive, with one exception. If the question has multiple intentions, it might expect more than one credibility type.

Subjective applies to questions where the expected answer is shaped by personal experiences, preferences, or interpretations. Such answers are not strictly bound by objective evidence. An example question is: "What is the common opinion about applying the layered architecture design pattern?" This type of question seeks insights that may vary across individuals or contexts.

Objective characterizes questions that require answers grounded in verified data, factual records, or empirical observations. The goal is to produce a response based on measurable, observable, or documented information, free from personal bias or interpretation.

Normative refers to questions in which the expected answer is informed by value-based judgments, recommendations, or ethical principles. These answers reflect what ought to be the case, drawing upon norms, societal values, or policy guidelines.

6.7.8. Question Goal

The **Question Goal** category classifies questions according to the primary objective they pursue in the scholarly literature search process. In most cases, only one class from this category should be applied. However, if a question contains multiple intentions, it may contain multiple goals, consequently requiring multiple classes for classification.

Class	Description
Information Lookup	Questions that seek factual information or descriptions about something.
Reasoning	Questions aimed at understanding why something occurs..
Problem Solving	Questions aimed aimed at identifying practical solutions, strategies, or methods to overcome a challenge or limitation.
Problematization	Questions aimed at articulating deficiencies or problems in current theories or practices.
Improvement	Questions aimed at enhancing existing tools, methods, or practices.
Prediction	Questions aimed aimed at anticipating future developments, outcomes, or trends.
Other Goal	Questions whose goal is not covered by any of the other classes.

Table 6.18.: The question goal category of the KGQA retrieval taxonomy. It classifies questions by the goal of the scholarly literature search process.

Information Lookup refers to questions that seek a presentation of factual information from the KG without requiring further explanation or justification. These are typically descriptive in nature. An example is: "What are the characteristics of an event-driven architecture?"

Reasoning applies to questions that seek to uncover causes, mechanisms, or theoretical explanations for observed phenomena. The goal is to understand why something happens without necessarily seeking a solution. For instance: "Why is the application of an event-driven architecture useful for improving maintainability?"

Problem Solving classifies questions that explicitly ask for practical solutions, strategies, or methods to address a stated problem. An example would be: "How can I apply the event-driven architecture pattern to improve maintainability?"

Problematization refers to questions that articulate issues or shortcomings in existing theories or practices. These questions often indicate the need for further scientific investigation. An example is "What difficulties do practitioners encounter when applying the event-driven architecture pattern?"

Improvement is used when the question aims to enhance existing tools, methods, or practices. It goes beyond problem identification by seeking actionable ways to make refinements. For example: "How can the maintainability of a software system be improved?"

Prediction classifies questions that attempt to anticipate future developments, trends, or outcomes. These are speculative in nature. An example is "How likely is it that LLMs will replace software developers in the future?"

Other Goal serves as a residual class for questions whose aims do not align clearly with any of the aforementioned classes.

6.8. Application

In this section, we document the APPLICATION phase of the taxonomy construction process. In the following, we first apply the created taxonomy to research questions to demonstrate the applicability of the taxonomy and to evaluate the last question **Q2.3** from the GQM plan in 5.1. Then, we explain where the taxonomy is intended to be applied. Finally, we present several guidelines for applying the taxonomy.

6.8.1. Application on Software Architecture Research Questions

To validate the practical applicability and descriptive power of the proposed taxonomy, we used research questions extracted from the ICSA/ECSA publication dataset that was used during our evaluation (see Section 7.3.2). To extract these research questions from the full text of these papers, the LLM-based extraction component of the SQA system (described in Section 7.1.1) was used. This process yielded a total of 231 research questions.

For our manual application of the taxonomy, a selection process was performed to reduce the total number of research questions to 20. The selection of a subset of these questions adhered to two primary criteria:

1. **Diversity of Origin:** Ensuring questions are sourced from a wide range of publications to capture varied research intentions.
2. **Random Selection:** Employing a random sampling method to minimize selection bias.

To implement this selection, a script was developed to perform stratified random sampling. This script utilized the classification of the publications according to [141] based on their *research level* and *paper class* to establish sampling categories. Publications lacking identifiable research questions were excluded prior to sampling. Subsequently, 20 questions were drawn uniformly across the categories using a random seed. The random seed was set to *2025*, the year of publication of this thesis, to guarantee the reproducibility of the selection process and to avoid introducing selection bias.

Following selection, the 20 research questions were manually classified according to our proposed taxonomy ($\mathcal{T}_3$). To address the validation question **Q2.3** regarding the significance and added value of the taxonomy, a comparative analysis was necessary against existing classification schemes. To the best of our knowledge, there is no standardized taxonomy

for categorizing questions in the KGQA field. Therefore, we determined that existing KGQA benchmark datasets, which incorporate question classifications, serve as the most appropriate reference. Consequently, the classification schemes from DBLP-QuAD [24] and LC-QuAD 2.0 [31] were selected for comparison. Although SciQA represents another potential source, its classification categories were deemed too dataset-specific for general applicability in this context and were therefore excluded.

Assumptions Regarding Underlying Knowledge Graph Structure The application and interpretation of the KGQA taxonomies inherently depend on the assumptions about the underlying KG structure that a KGQA system would query to answer classified questions. To apply the taxonomies, it is essential to understand the structure of the underlying KG on which the questions are based. Therefore, we made the following assumptions for the graph that contains the relevant data:

1. **Granularity of Representation:** The KG is assumed to store information in semantically distinct nodes, facilitating a fine-grained representation. As a result, addressing the majority of questions requires retrieving and possibly integrating information from several nodes unless the question specifically refers to a single piece of atomic data that can be encapsulated within a single node.
2. **Absence of Precomputation:** It is assumed that answers, particularly for complex questions such as aggregation, comparison, or counting, are not precomputed and stored within single nodes. Instead, deriving such answers involves retrieving and integrating information distributed across multiple nodes at query time.

These assumptions align with common practices in the design of KG in which complex facts are decomposed. This implies that the complexity of the question, as captured by the taxonomy, correlates with the complexity of the required graph traversal and data integration operations.

6.8.1.1. Validating the Significance

The question **Q2.3** from the GQM plan provided in Section 5.1, investigates whether the proposed KGQA retrieval taxonomy offers a significant improvement in descriptive precision compared to existing schemes. This is evaluated using the *classification delta* metric (**M2.3.1**), which quantifies the difference in classification granularity. A manual classification of the 20 selected research questions was performed using our taxonomy ($\mathcal{T}_3$) and the categories provided by DBLP-QuAD [24] and LC-QuAD 2.0 [31]. As such, we measure $classification\_delta(\mathcal{T}_3, \{\text{DBLP-QuAD, LC-QuAD 2.0}\}, C) = \frac{8}{15} = 0.533$. This indicates that applying $\mathcal{T}_3$ resulted in the use of 15 distinct classifications from our taxonomy to classify the 20 questions. In contrast, applying the classification schemes from DBLP-QuAD and LC-QuAD 2.0 to the same set of questions resulted in the use of eight distinct classifications. According to the metric **M2.3.1**, this demonstrates that our taxonomy $\mathcal{T}_3$ offers substantially higher granularity, enabling a more precise description and differentiation of the research questions based on their KGQA retrieval characteristics.

Moreover, we found that the classifications from DBLP-QuAD and LC-QuAD 2.0 possess a limited number of categories, leading to coarse-grained classifications that often fail to adequately distinguish the varying structural and semantic complexities in the research questions. Furthermore, the lack of clear hierarchical structuring or concern-based categorization within these schemes hinders the systematic analysis of questions.

These quantitative and qualitative findings support the conclusion that the proposed KGQA retrieval taxonomy addresses a notable gap. This is because it provides a structured and fine-grained framework for classifying questions intended for KGQA systems, with each category addressing distinct retrieval characteristics and potential challenges. We expect such a classification to be valuable not only for understanding the nature of questions but also for assessing the capabilities and limitations of KGQA systems.

6.8.1.2. Discussion on Applicability on Software Architecture Research Questions

After applying our KGQA taxonomy to research questions in the SWA domain, we discuss our findings here.

To begin with, we were able to apply all proposed categories of the taxonomy to all selected research questions. This was expected, as we ensured during the creation of the taxonomy that it could be generally applied regardless of the underlying domain. However, we observe that all the questions are classified as *objective* in the *Answer Credibility* category. This is presumably because research questions posed in SWA are predominantly empirical in nature. Consequently, the expressiveness of this particular category for classifying research questions in SWA is limited.

Furthermore, it is important to note that our taxonomy, due to its chosen high level of abstraction, does not classify the specific semantic content of a question. Instead, it focuses on the type of expected answer and the stated constraints. Therefore, if more detailed content-based classifications are desired, the categories *Answer Type* and *Condition Type* might need to be adjusted to incorporate domain-specific classes. For this purpose, within the context of SWA, the classifications by Shaw [125] and Easterbrook et al. [126, pp. 287–290] prove helpful, as they propose categorizations of research questions specifically for SE. In the following, we discuss their relationship to our taxonomy.

Shaw [125] categorizes SE research questions by their Type of Questions and Type of Results. The question types proposed by Shaw, such as seeking a *Method or means of development* or aiming for *Generalization or characterization*, generally align with our *Question Goal* category. For example, a question concerning a new development method implies a *Problem Solving* goal in our taxonomy, while a question about generalization or characterization aligns with the *Information Lookup* or *Reasoning* goal. Similarly, the Type of Results dimension by Shaw, such as *Procedure or technique*, *Qualitative or descriptive model*, or *Tool or notation*, correspond to our *Answer Type* category and further inform the expected *Answer Format*. A *Tool or notation* would be classified as a *Named Entity*, while a *Procedure or technique* or a *Qualitative or descriptive model* could be classified as either a *Named Entity* or a *Description*. The answer format would likely be *Explanatory* or *Enumerative*. We recognize

that, depending on the use case, extending the *Named Entity* class with further subclasses derived from the work of Shaw could be beneficial. Nevertheless, while Shaw frames research objectives from a SE perspective, our taxonomy approaches this classification from the viewpoint of an interaction with a KGQA system.

Easterbrook et al. [126] focus on the kind of knowledge sought, particularly distinguishing between *Knowledge questions* and *Design questions*. The various *Knowledge questions* identified by Easterbrook et al., such as *Exploratory questions*, *Relationship questions*, and *Causality questions*, find parallels in our taxonomy. For example, *Exploratory questions* that ask for existence often expect a *Boolean* or *Named Entity Answer Type*, while those asking for a description map to the *Description* class. Questions that focus on relationships or causality align directly with our *Relationship* class within the *Retrieval Operation* category. In contrast to these knowledge-based questions, Easterbrook et al. also identify *Design Questions*. These align well with our *Problem Solving* or *Improvement* goals, with the expected *Answer Type* typically being a *Description* of a method or strategy.

In the SWA setting, the primary value of our KGQA retrieval taxonomy lies in its distinct focus on the operational aspects of knowledge retrieval from a KG. Although Shaw [125] and Easterbrook et al. [126, pp. 287–290] define the *kind* of knowledge sought within the research domain, our taxonomy clarifies *how* answers to such questions might be structured and retrieved from the KG. This focus, in turn, aids in understanding the complexities involved in the retrieval process. We conclude that our taxonomy maintains a high level of abstraction, ensuring broad applicability. We anticipate that in scenarios where the specific nature of the knowledge is a priority, the categories *Answer Type* and *Condition Type* would need to be specialized to include terms specific to the applied domain.

6.8.1.3. Illustrative Examples of Research Question Classification

This section presents two illustrative examples of the classification of research questions using the proposed taxonomy. Each example includes the question, its classification across the defined categories, and a justification for these classifications. The complete list of the research questions on which we applied the taxonomy is provided in the replication package [124].

Question: "What is the advantage of Butterfly Space modeling in identifying performance optimization opportunities compared to dynamic profiling?" [142]

Answer Type: Description

Condition Type: Description, Named Entity

Answer Format: Explanatory

Retrieval Operation: Comparison

Graph Representation: Multi Fact

Intention Count: Single Intention

Question Goal: Information Lookup

Answer Credibility: Objective

The question asks, "What is the advantage..." implying that the answer should explain the benefit of Butterfly Space modeling relative to dynamic profiling in a specific context. Consequently, the *Answer Type* is classified as description, and the corresponding *Answer Format* is explanatory. Furthermore, the question imposes several conditions "Butterfly Space modeling", "dynamic profiling", and "identifying performance optimization opportunities". These elements represent both named entities and descriptive constraints, leading to the classification of the *Condition Type* as description and named entity. The use of "compared" clearly indicates that the required *Retrieval Operation* involves a comparison.

Based on the previously defined assumptions regarding the knowledge graph structure, the question is classified as multi fact under *Graph Representation*, as answering it requires synthesizing information distributed across multiple graph entities. Moreover, the question possesses a singular, clear objective focused on comparison, which cannot be decomposed into independent subquestions without losing semantic integrity. Therefore, the *Intention Count* is single intention. The *Question Goal* is information lookup, as the inquiry aims to retrieve factual information regarding the specific advantage, seeking knowledge about "how something is". Finally, expected answers are anticipated to rely on verifiable data or empirical evidence rather than subjective viewpoints, classifying the *Answer Credibility* as objective.

Question: "What dimensions impact how interfaces in agile automotive contexts are changed and how are the dimensions related?" [143]

Answer Type: Description, Named Entity

Condition Type: Description

Answer Format: Enumerative, Explanatory

Retrieval Operation: Aggregation, Relationship

Graph Representation: Multi Fact

Intention Count: Multiple Intentions

Question Goal: Information Lookup

Answer Credibility: Objective

The question includes two types of intentions since it asks both for "What dimensions..." and "how are the dimensions related?". This structure signifies multiple underlying objectives, classifying *Intention Count* as multiple intentions. For *Answer Type*, the request for "What dimensions ..." points to the type of named entities that represent identifiable factors. The subsequent request for *how* these dimensions relate requires a descriptive explanation in natural language. Therefore, the classification includes both the description and the named entity. The specified condition "agile automotive contexts" is descriptive rather than a

named entity, as it requires the retriever to understand whether an interface resides in the given context. Therefore, *Condition Type* is a description.

The *Answer Format* combines enumerative and explanatory aspects. An enumerative format is required to list the identified dimensions, while an explanatory format is needed to detail the relationships between them. Correspondingly, the *Retrieval Operation* involves both aggregation to collect the dimensions and relationship analysis to understand their connections. Consequently, the *Graph Representation* requires multiple facts.

The *Question Goal* is classified as information lookup because the question focuses on identifying and understanding existing knowledge. Lastly, the expected answer necessitates grounding in research findings or empirical evidence, distinguishing it from subjective opinion. Consequently, the *Answer Credibility* is classified as objective.

6.8.2. Potential Applications of the Taxonomy

The proposed taxonomy offers a structured framework that is applicable to several contexts related to the formulation of questions, the retrieval of information and the design of the system, particularly in the domain of KGQA and the search for scholarly information. We consider the taxonomy to be helpful in various activities:

- **Design and Development of KGQA Approaches:** The taxonomy provides a detailed characterization of the question types that a KGQA system encounters. Therefore, the application of the taxonomy can help guide the architectural design, selection of appropriate retrieval algorithms, and implementation of specific functionalities to handle diverse questions effectively.
- **Evaluation of KGQA Approaches:** The taxonomy allows the creation of standardized evaluation datasets. By classifying questions according to the categories provided it allows for comparing the performance across different types and complexities of the questions to understand the capabilities and limitations of a KGQA system.
- **Knowledge Graph Engineering:** Applying the taxonomy to questions frequently posed against a KG can provide valuable feedback for KG design and optimization. This is because the taxonomy can highlight the need for specific schema structures, precomputation of certain relationships or aggregations, or adjustments to data granularity to better support common query patterns posed to KGQA retrieval systems.
- **Guiding Question Formulation:** The taxonomy may also be helpful in the creation of question formulations that address various aspects of KGQA. Awareness of the categories provided by the taxonomy can be used to improve the precision and effectiveness of the questions for successful and targeted information retrieval.

6.8.3. Guidelines for Applying the Taxonomy

To ensure a consistent, accurate, and meaningful application of the taxonomy, we provide the following guidelines:

- **Systematic Category Evaluation:** Each of the categories provided should be systematically addressed one by one.
- **Knowledge Graph Structure Awareness:** Accurate classification of questions within the *Graph Representation* and *Retrieval Operation* categories necessitates familiarity with the underlying structure of the graph. It is therefore crucial to determine whether required information corresponds to single or multiple triples and whether complex operations are precomputed within the graph or must be performed at query time.
- **Anticipation of Expected Answers:** For the classification of the *Answer Type* and *Answer Format* categories it is helpful to formulate an expected answer. The inherent types and structure of this answer can then be used to determine the appropriate classes.
- **Distinguish Conditions from Answers:** When classifying according to the *Condition Type* category, it is important to focus on the constraints explicitly stated within the provided question itself. This differs from the characteristics of the expected answer, which are covered by *Answer Type*.
- **Specificity in Retrieval Operation:** The category *Retrieval Operation* includes some classes that subsume others. For example, *Superlative* or *Ranking* may involve counting or aggregation. For a consistent classification, assign the class that represents the most complex or defining operation required to answer the question.
- **Intention Count Criterion:** To understand whether a question should be classified as having multiple intentions, a *splitting* test can be helpful. Here, it should be determined whether the question can be broken down into distinct and independently meaningful subquestions without losing the original overall meaning.

6.9. Threats to Validity

In the following section, we discuss the threats to validity of the second contribution **C2** (the KGQA retrieval taxonomy). This extends the limitations of the taxonomy construction methodology, which we discuss in Section 5.4. The structure of the subsequent threat evaluation follows the classification framework suggested by Runeson and Höst [144], which encompasses the concepts of construct validity, internal validity, external validity, and reliability.

Construct Validity Construct validity addresses whether the measurements applied truly represent the evaluation objectives. Fundamentally, during the validation of the taxonomy, we employed a GQM plan to minimize the risk that the created increments of the taxonomy fail to align with the defined goals (**G1 and G2**). To facilitate this, we used the evaluation method from Kaplan et al. [82], which provides a comprehensive framework for the validation of taxonomies. Additionally, we interpreted the results of the evaluation goals not just based on individual metrics but also critically questioned the results to consider the impact on the overall objective. This critical evaluation aimed to reduce the risk of drawing conclusions inconsistent with the intended goals based solely on metric values.

Internal Validity Internal validity examines whether the results depend on other factors or are solely attributable to the observed factors. The primary threat to validity here is the selection of literature used to create the taxonomy. There is a risk that insufficient literature was considered and that adding more literature could have introduced additional classes for the taxonomy creation. This threat was mitigated through several measures: 1) supplementing the initial seed-based search with broader queries on academic search engines like Google Scholar, 2) categorizing identified papers by domain and topic to ensure a diverse input, and 3) thoroughly documenting the entire search process and outcomes in structured artifacts (e.g., JSON files), enhancing transparency and facilitating future extensions.

Another threat is the inclusion of literature beyond the primary domain of KGQA. We argue that this broad inclusion was necessary because we did not find sufficient resources within the KGQA field for a broad classification. This introduces the risk of incorporating concepts or structures irrelevant to the specific application context of KGQA retrieval. This risk was primarily addressed during the Relevance Assessment phase, where the extracted and clustered concepts were explicitly evaluated against the defined objectives and scope of the KGQA retrieval taxonomy. Consequently, concepts deemed insufficiently relevant to this specific context were excluded.

External Validity External validity concerns the generalizability of the findings, specifically the extent to which the developed taxonomy can be applied beyond the immediate context of its creation. The taxonomy was specifically designed to classify aspects related to literature search tasks within the KGQA domain. However, the process required incorporating insights from the literature covering various application areas due to the limited availability of dedicated KGQA classification studies. Consequently, certain structural elements or categories within the taxonomy are likely to capture fundamental aspects that are not exclusive to the intended use case. Therefore, parts of the taxonomy might have relevance for other research domains or information retrieval contexts. However, determining the precise extent of this generalizability would require further empirical validation in different settings.

Reliability Reliability addresses the consistency and repeatability of the study, specifically whether the outcomes depend on the specific researchers involved. The primary threat to reliability in this work stems from the fact that the taxonomy development process,

particularly its subjective phases, was conducted by a single researcher. Key steps involving subjective judgment include: 1) filtering literature based on inclusion/exclusion criteria, 2) assessing semantic similarity for class deduplication and clustering, 3) evaluating the relevance of categories and classes, and 4) making refinement decisions based on quantitative and qualitative evaluations. There is a possibility that if different researchers were to replicate the process, the results might vary. This threat was mitigated by: 1) documenting the entire procedure in detail, including the reasons for key decisions, both in this thesis and the associated replication package artifacts to improve transparency and traceability. 2) Operationalizing the process and providing structured artifacts in JSON and BibTeX format facilitates replication and allows other researchers to analyze, adapt, or extend the taxonomy based on alternative judgments or additional evidence. Although subjectivity cannot be entirely eliminated, these measures aim to maximize transparency and the potential for independent verification or refinement of the results.

7. Implementation

In this chapter, we outline the key implementations that we implemented to conduct the evaluation in Chapter 10. We begin in Section 7.1 by detailing the implementation of the SQA framework, which forms the basis for all subsequent implementations and the realization of the experimentation setup. Then, Section 7.2 presents the specific implementation details of our proposed HubLink approach (**C1**). Following this, Section 7.3 describes the implementation aspects related to the integration of the ORKG within the SQA framework. Last, Section 7.4 provides details on the implementation of the various KGQA baseline methods used for comparative evaluation against our proposed approach.

7.1. Scholarly Question Answering Framework

To effectively demonstrate and evaluate the performance of the HubLink retriever within the scholarly literature search task, we developed a dedicated Python framework termed as Scholarly Question Answering (SQA) system. This framework provides an environment specifically designed for the systematic testing and evaluation of diverse KGQA approaches in academic domains.

In the following sections, we briefly introduce each component of the SQA system. We first introduce the key capabilities that the SQA framework provides. Then, we outline the overall system architecture, including the different components that the framework consists of. Subsequently, each component of the framework is described in detail.

7.1.0.1. Capabilities

The SQA framework is equipped with a range of capabilities designed to ensure flexibility, reproducibility, and comprehensive evaluation possibilities for KGQA research. These features collectively provide a powerful toolkit for constructing, executing, and analyzing RAG pipelines tailored to scholarly information needs. Key capabilities include:

[C1] **Flexible Configuration Management:** Utilizes easily modifiable JSON-based configuration files to define and serialize parameters for all system components. This ensures straightforward reproducibility and systematic modification of experiments.

[C2] **Experiment Execution and Evaluation:** Automates the process of conducting experiments using pipelines defined in configuration files. It evaluates performance using a suite of relevant metrics (e.g., retrieval recall, answer relevance, factuality), stores detailed outcomes along with reproducibility information, and facilitates results visualization through generated diagrams for easier analysis.

[C3] **Data Ingestion:** Provides modules for loading publication data and QA pairs from standard JSON and CSV formats, serving as the foundation for knowledge base creation and evaluation datasets.

[C4] **Modular RAG Pipeline:** Implements a fully customizable RAG pipeline architecture comprising pre-retrieval, retrieval, post-retrieval, and generation stages. This modularity allows easy interchanging, configuration, and testing of different algorithms or models at each stage of the workflow.

[C5] **Scientific Text Extraction:** Integrates functionality to extract structured information from the text of publications by leveraging an LLM.

[C6] **Knowledge Graph Integration:** Supports the modular construction and integration of diverse KGs by providing a unified interface.

[C7] **Semi-Automated KGQA Pair Generation**: Incorporates strategies such as graph clustering and subgraph extraction to assist in the generation of relevant QA pairs directly from the underlying KGs.

[C8] **Command-Line Interface (CLI)**: Offers a CLI application for managing configuration files, triggering data ingestion, executing individual experiments, and conducting interactive QA sessions.

7.1.0.2. System Architecture

The SQA framework is designed with extendability in mind. Consequently, it provides well-defined abstract interfaces and base classes to facilitate the straightforward integration of new KGQA approaches, new KGs or vector databases, different LLMs, and custom intermediate pipeline stages, ensuring the framework can adapt to future research directions and technological advancements. The framework employs a modular software architecture, as depicted in Figure 7.1 to promote a clear separation of concerns, where each component encapsulates a specific set of functionalities. In the following, we briefly introduce each component of the SQA framework before explaining them in detail:

Configuration Component The initialization and experimental setup are fundamentally based on the *Configuration* component, which parses and validates JSON-based configuration files that define parameters and selected modules for all other components.

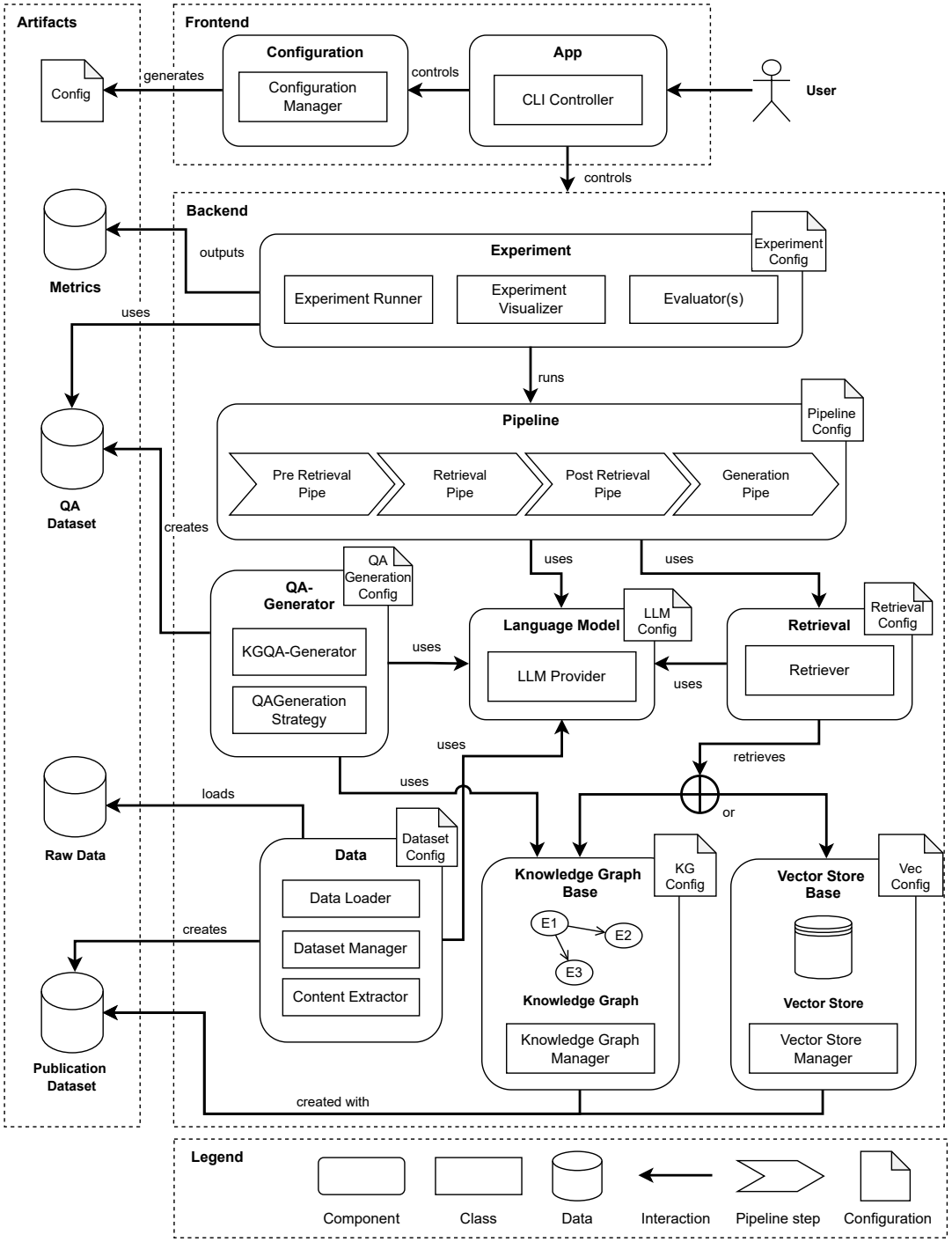


Figure 7.1.: Scholarly question-answering system architecture, illustrating key components and their primary interactions

Data Component Data loading, preprocessing and distribution are the responsibility of the data component. It manages metadata and content data from publications, as well as QA pairs from JSON and CSV sources, and makes these accessible to other parts of the system.

Pipeline Component The core logic of the question-answering process resides within the *Pipeline* component. This component orchestrates the sequence of pipeline operations (pre-retrieval processing, context retrieval, post-retrieval processing, and final answer generation).

Retriever Component Integrated closely with the pipeline, the *Retriever* component performs the task of searching the designated KB (KG or vector index) to find contextually relevant information passages or facts relevant to the input question.

Language Model Component A unified interface for interacting with local or API-based LLMs is provided by the language model component. It handles LLM access for other components of the system and parses the response.

Knowledge Base A KB stores and provides access to the data in the system. In the system, two types of KBs are supported, which are KGs and vector indices. Both are unified in the knowledge base component.

Experiment Component The experiment component provides the functionality to perform systematic evaluations. It iterates over specified configurations, executes the QA pipeline for each test question, collects results, computes relevant RAG metrics, and stores these findings for later analysis.

QA Generator Component This component helps to create evaluation datasets by implementing semi-automated methods to generate KGQA pairs from a KG using an LLM.

App Component User interaction is managed through the app component, which offers a CLI application. This tool allows users to manage configurations, execute experiments, ingest data, and perform interactive querying of pipelines.

7.1.1. Data Component

The SQA system processes several types of data and features the structured LLM-based extraction of text. This section details the primary inputs and outputs handled by the system and describes the process of extracting structured data from paper texts.

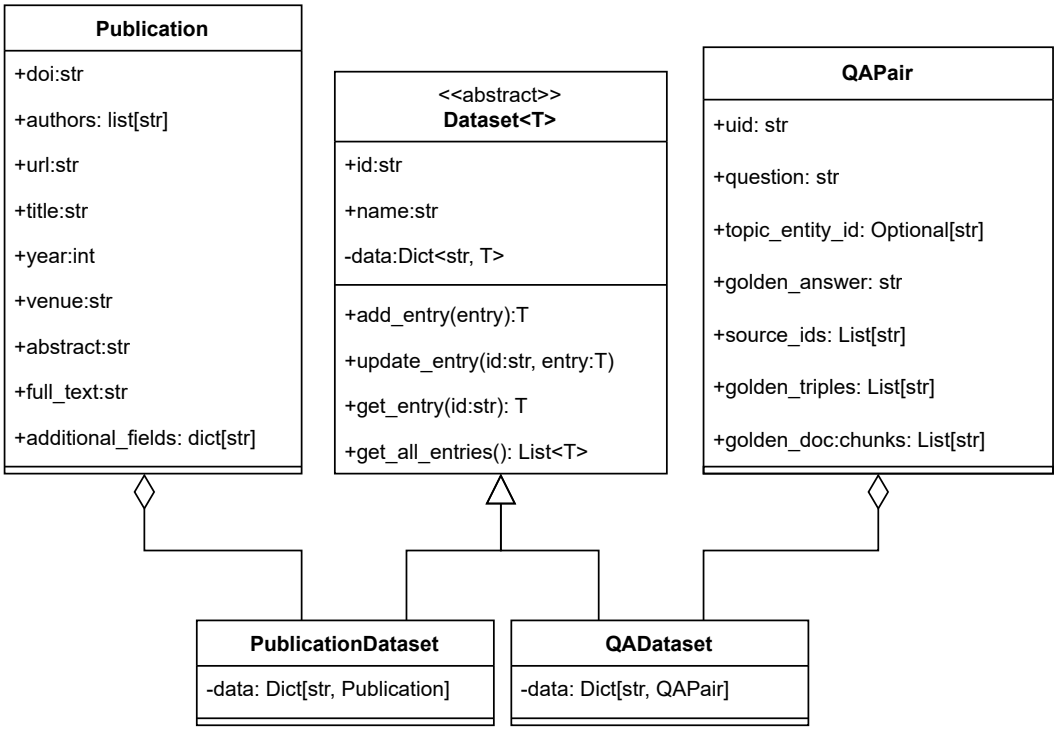


Figure 7.2.: Input data models of the SQA system.

7.1.1.1. Inputs

Any data that is inserted into the system has to be converted to either a `Publication` or `QAPair` object as depicted in Figure 7.2. We introduce them in the following:

Publication Data: An important data input of the SQA framework is the metadata and content texts of scientific publications. These are ingested into the system from raw data, for example, in JSON format. For our experiments, we implemented a dedicated `JsonDataLoader`, which loads the publication data and converts it into the internal `Publication` data model that encapsulates both metadata and available full text for each scientific publication. To handle multiple publications, the `PublicationDataset` data model is used. This data model stores a collection of publications and provides easy access to them. The dataset is managed by the `DatasetManager` class, which enables other components to access datasets by providing configuration files. The manager then initiates the loading and handles the persistence.

Question-Answer Pair Data: Question-answer pairs include the questions and ground truth to conduct the experiments. These are internally stored as `QAPair` objects in the SQA system and loaded from data loader implementations from raw formats such as CSV. For our experiments, we implemented a `CSVDataLoader` that allows the serialized KGQA dataset

to be imported and used for the experiments. The `QAPair` data model has the following attributes:

1. **UID:** A unique identification identifier that uniquely identifies the specific question-answer pair.
2. **Question:** The natural language question posed to the QA system.
3. **Topic Entity:** An optional field specifying an entity within the target KG, which potentially serves as an entry point for certain retrieval algorithms. It is optional as not all retrievers require this information.
4. **Golden Answer:** A reference answer to the question, formulated in natural language, considered correct for evaluation purposes.
5. **Golden Triples:** A set of triples present in the target KG that represent the factual basis for the *Golden Answer*.
6. **Golden Text Chunks:** The text passages from source publications containing information that correspond to the *Golden Answer* and align with the *Golden Triples*.
7. **Source IDs:** Identifiers of the source publications from which the *Golden Answer*, *Golden Triples*, and *Golden Text Chunks* were derived.

Similarly to the publication data, a collection of `QAPair` is handled by a `QADataset` data model. This model handles the access to the pairs and their persistence.

7.1.1.2. Outputs

Depending on the executed workflow (e.g., experimentation, interactive querying, QA generation), the SQA system produces several outputs:

Experiment Results: When running an experiment, for each question several outputs are produced. These outputs include the generated answer and the retrieved context that both relate to the provided question. These outputs are then evaluated against the golden ground truth data provided by the `QADataset` using dedicated `Evaluator` objects. These produce comprehensive evaluation reports containing various RAG metrics (e.g., context relevance, answer faithfulness) and other measurements, which are then aggregated and exported in `CSV` format. Subsequently, an `ExperimentVisualizer` can be used to generate plots and tables to visualize the results.

Simple Answer: When executing a RAG pipeline in interactive mode, the primary output returned to the user is the generated natural language answer to the question posed.

QA Generation Data: Another output comprises the semi-automatically generated `QAPair` objects, which are stored in a `QADataset` structure and serialized into `CSV` format for subsequent use in evaluations.

7.1.1.3. Data Extraction

The SQA system also provides the ability to extract scientific content directly from the text of scientific publications. This process, orchestrated by the `ContentExtractor` class using a configured LLM, transforms unstructured text into structured data suitable for enriching KGs. The extraction process involves the following steps:

1. **Text Chunking:** The source text of the publication is segmented into smaller, manageable chunks suitable for processing by the context window limits of the LLM.
2. **Chunk Extraction:** Each text chunk is processed by the LLM. To enhance completeness, this extraction is done multiple times for the same chunk. In subsequent extractions for the same chunk, previously extracted information is aggregated and provided back to the LLM as context to minimize redundancy.
3. **Tracing:** Each extracted data item is mapped back to its location in the original publication source. This trace provides provenance for the information extracted.

As mentioned above, the core extraction mechanism involves prompting the LLM. This is accomplished by populating a predefined `JSON` schema which defines the target information types, attributes, and relationships to be extracted (e.g., research methods used, research questions, key findings). The task of the LLM is to analyze the provided text chunk and accurately populate this schema with corresponding information found within the text. The data extracted from the schema can subsequently be utilized, for instance, in the construction of a KG.

7.1.2. Configuration Component

A primary goal during the development of the SQA system was to ensure the easy definition and reproducibility of experiments. To achieve this, the system utilizes configuration files stored in `JSON` format. This storage method serializes configurations, enabling their persistence for future use and allowing experiments to be repeated accurately by using the corresponding files.

Each component of the SQA system is controlled through a dedicated configuration file that defines its required parameters. An important characteristic of a configuration is that it can be hierarchically defined, meaning that one configuration may include another. Figure 7.3 shows the KG configuration model as an example. It illustrates the parameters required, demonstrating that configurations can contain both primitive data types, such as strings and integers, and complex structures, like nested configuration objects.

To load the JSON files into the SQA system the `ConfigurationManager` is responsible. This manager interprets the file to confirm that it is correctly structured and provides the filled configuration object to the relevant component that needs to be initialized. Furthermore, for reproducibility and caching, it is essential to determine whether two configurations are identical. This is achieved by calculating a cryptographic hash value for each configuration

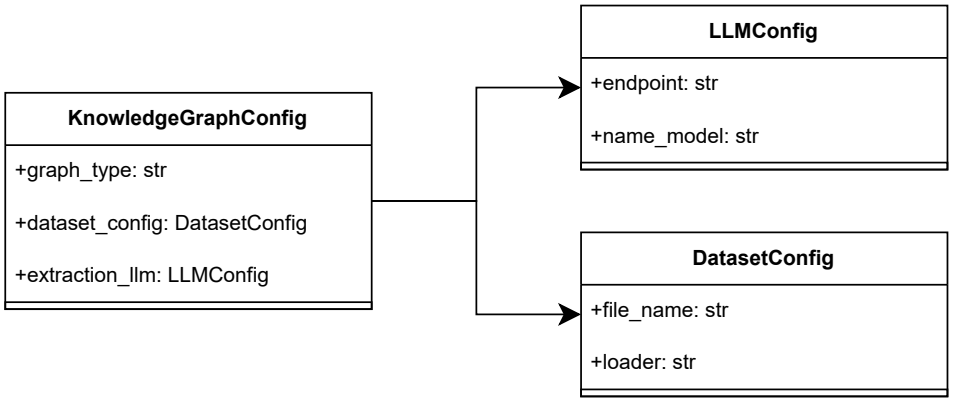


Figure 7.3.: The configuration model for a Knowledge Graph in the SQA system.

object. By calculating this hash over the content of the configuration, we ensure that the values of two configurations are only identical if and only if their contents are identical. This mechanism makes it easy to determine whether two configurations are the same. This functionality is used by components in the system to avoid duplicate instantiation in caching scenarios such as storing LLM models or KGs.

7.1.3. Pipeline and Pipes Component

The RAG process is implemented within the SQA system using a pipeline architecture facilitated by the LangChain library[1]. This pipeline accepts a question and, optionally, a topic entity as input. The pipeline processes this input and returns a `PipeIO` object populated with the results. The `PipeIO` object acts as a data container, progressively accumulating information such as the retrieved contexts and the generated answer as it traverses the pipeline stages. It also collects tracking data like carbon emissions or LLM token usage.

In the following sections, we first describe the retrieval pipeline. Then, we detail the `PipeIO` data model.

7.1.3.1. Retrieval Pipeline

A standard pipeline comprises four distinct stages, referred to as *pipes*. These pipes are executed sequentially in a predefined order: Execution starts with the *Pre-Retrieval Pipe*, followed by the *Retrieval Pipe*, then the *Post-Retrieval Pipe*, and concludes with the *Generation Pipe*. These pipes are explained in the following:

Pre-Retrieval Pipe: This pipe is responsible for processing the input question before retrieval takes place. For example, the question given as input into the pipeline can be expanded to include synonyms or related terms to improve recall.

[1] `https://www.langchain.com/` [last accessed on 12.05.2025]

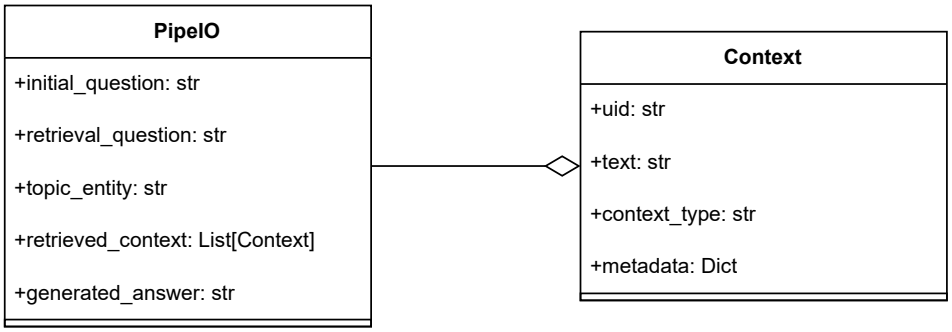


Figure 7.4.: The UML Diagram of the PipeIO model.

Retrieval Pipe: This is the pipe that is responsible for the extraction of relevant contexts from the underlying KB. All retrievers are applied within this pipeline step. The task of the retriever is to identify and retrieve relevant context passages or structured data from the designated KB. The retrieved contexts are then stored in the `PipeIO` object. In addition, certain retrievers may generate a final answer directly. If the retriever does not generate an answer, a subsequent *Generation Pipe* is required to generate the answer.

Post-Retrieval Pipe: This optional stage processes the contexts retrieved in the previous stage. Common operations include filtering irrelevant passages, reranking of contexts based on relevance or other criteria, or selecting a subset of contexts based on a specified criteria.

Generation Pipe: This pipe is responsible for synthesizing the final natural language answer. A configured LLM is invoked, provided with the original question and the retrieved contexts. The task of the LLM is to generate a coherent and factually grounded answer based on this input. The generated answer is then stored in the `PipeIO` object, completing the pipeline execution. This stage is optional if the retriever already produced a final answer.

7.1.3.2. PipeIO Datamodel

The data model of the `PipeIO` object is shown in Figure 7.4. It contains several attributes that are populated by the individual pipes in the pipeline. Initially, the `PipeIO` data model is created by the pipeline, where the attributes `initial_question`, `retrieval_question`, and `topic_entity` are initialized. The distinction between these two types of question exists because when the question is modified by the pre-retrieval pipe, the original question should still be saved. Consequently, the `initial_question` attribute is `read-only`, while `retrieval_question` can be modified. Furthermore, the topic entity can optionally be added. This allows retrievers to have an entry point in the KG if they require it. Moreover, the `PipeIO` object contains the attributes `retrieved_context` and `generated_answer`. These attributes are filled during execution.

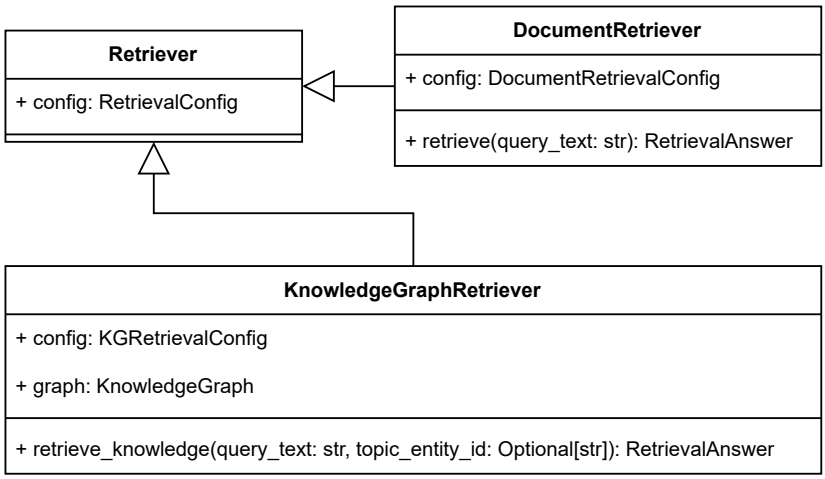


Figure 7.5.: The UML diagram for retrievers in the SQA framework.

7.1.4. Retriever Component

The main class in the retrieval component is the `Retriever`, which is responsible for finding relevant contexts from a KB to answer a question. There are two sub-classes available for the `Retriever`: the `DocumentRetriever` and the `KnowledgeGraphRetriever`, each handling different types of data:

Document-based Retriever works with text documents as input sources, specifically the full text of scientific publications. Unlike the KG variant, the `DocumentRetriever` is not bound to a specific KB. Instead, the corresponding `DocumentRetrievalConfig` specifies the configuration of the dataset, which contains the data that should be used for retrieval instead of a KG.

Knowledge Graph-based Retriever operates directly on a KG. Instead of documents, it uses a KG as the underlying data store. The configuration for the KG is provided with the `KGRetrievalConfig` during the initialization of the retriever.

Once the retriever is implemented in the system, it is used in a pipeline to allow the retrieval of context for questions and to generate answers.

7.1.5. Language Model Component

Since many components of the SQA system work with LLMs, a dedicated *Language Model* component is provided. This component offers a unified interface to make requests to an LLM. The requests, in conjunction with their corresponding configuration, are handled by the `LLMProvider`, which is responsible for initializing the model and establishing the connection.

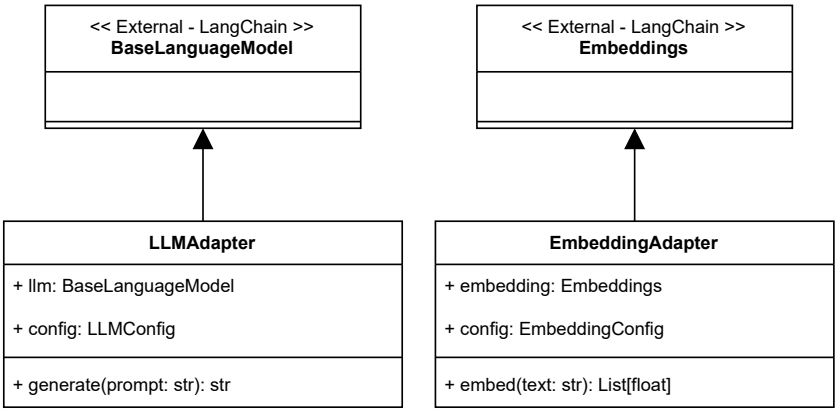


Figure 7.6.: The UML diagram for the LLM adapter and Embedding adapter classes of the SQA system.

The implementation of the models is achieved through the adaptation of the `BaseLanguageModel` and `Embeddings` classes from the LangChain[2] library. The advantage of using the LangChain framework lies in its support for a wide range of models, a well-maintained interface, and easy integration of new models. Two types of adapters are implemented, which are shown in Figure 7.6. The `LLMAdapter` is responsible for sending requests to the LLM and processing the responses. The `EmbeddingAdapter`, on the other hand, handles the transformation of texts into vectors.

7.1.6. Knowledge Bases Component

The SQA system provides two different types of KBs as shown in Figure 7.7: KGs and vector stores. These KBs can be used by retrievers to find relevant contexts for answering questions. In the following, we detail their implementations in the SQA system:

Knowledge Graphs are represented as RDF graphs in the SQA system. These graphs consist of triples represented with the `Triple` data model, which consists of `Knowledge` objects as shown in Figure 7.7. Moreover, to manage and provide `KnowledgeGraph` objects in the SQA system, the `KnowledgeGraphManager` is responsible. This manager initializes the graphs based on the provided configuration and caches their connection to allow other components to query the graph. The preparation or creation of a KG is facilitated by the `KnowledgeGraphFactory`. This factory receives the graph configuration file and initializes or creates the KG based on the given parameters.

Vector Stores are specialized data structures that enable efficient vector storage and querying. They are used to store texts that have been transformed into a low-dimensional vector

[2] `https://python.langchain.com/docs/integrations/chat/` [last accessed on 28.01.2025]

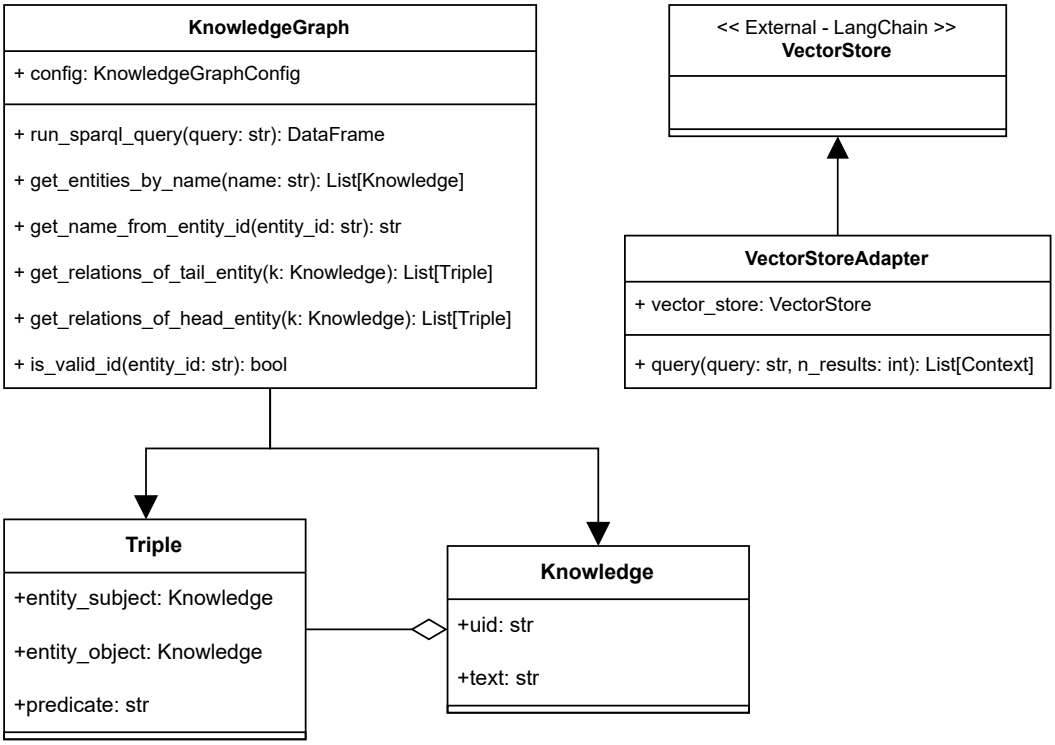


Figure 7.7.: The UML diagram for the knowledge graph and vector store.

space using an embedding model. These vectors can then be used to calculate the similarities between texts. In the SQA system, the `VectorStore` implementation from the LangChain library is adapted using a `VectorStoreAdapter`. Similarly to knowledge graphs, a `VectorStoreManager` is responsible for initializing, managing, and providing the vector stores.

7.1.7. Experiment Component

One of the main features of the SQA system is the execution of experiments. These are defined using configuration files as described in Section 7.1.2. The UML diagram of an `ExperimentConfig` is shown in Figure 7.8. This configuration contains all parameters necessary for conducting an experiment:

- **Base Pipeline Config** is the configuration of the pipeline used for answering questions. It contains all the necessary parameters to initialize an RAG pipeline.
- **Parameter Ranges** are a list of parameters that allow varying the pipeline parameters within an experiment, which is useful for studying the effects of parameters on the results. The parameter configurations specified here are used by the `ExperimentRunner`

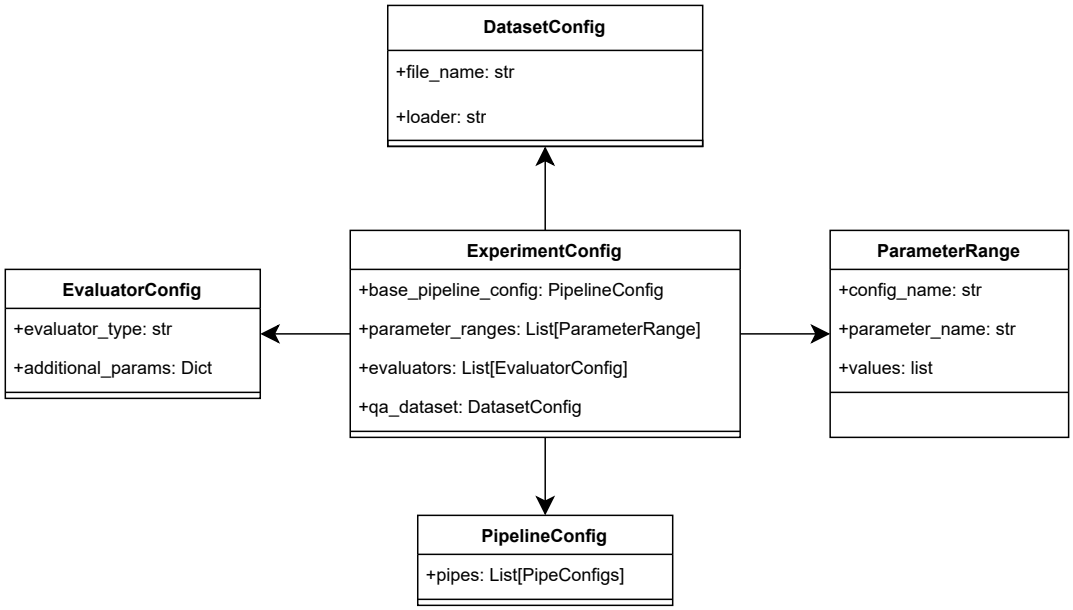


Figure 7.8.: The UML diagram for the experiment configuration

to create multiple pipelines based on the base pipeline configuration, which are then executed in batch.

- **Evaluators** is a list of configurations for evaluators. These are classes in the SQA system responsible for creating RAG metrics for the experiment.
- **QA Dataset** is the configuration of the `QADataset` used for the experiment. This contains the questions to be answered in the experiment, along with the corresponding ground truth data for evaluation.

The `ExperimentRunner` class is responsible for executing the experiment based on a configuration. This class takes the configuration and prepares the pipelines, which are then executed sequentially to fill the `PipeIO` object with data. Subsequently, the results of the pipeline are evaluated using the evaluators, which are also prepared by the runner. After the experiment has been conducted, the `ExperimentVisualizer` class is responsible for visualizing the results in diagrams and tables.

7.1.8. Question Answering Generator Component

This component is responsible for the semi-automated creation of KGQA pairs, which are stored internally as `QAPair` data models. The SQA framework implements two different strategies for generating these `QAPair` objects: the *Clustering Strategy* and the *Subgraph Strategy*.

In the following, we first describe the clustering and subgraph strategy. Then, we explain the context and answer validation of the generated questions and answers.

7.1.8.1. Clustering Strategy

A significant challenge in generating KGQA datasets is ensuring the completeness of the ground truth, particularly when an answer corresponds to multiple triples in the graph [129]. Ideally, the ground truth associated with a question should encompass all valid, relevant triples from the KG. Incomplete ground truth can lead to misleading evaluations where systems are penalized for retrieving valid facts not included in the reference set. The *Clustering Strategy* aims to address this challenge by identifying and grouping all potentially relevant facts before generating the question. The strategy operates as follows:

1. **Provide Parameters:** The strategy is initialized with various parameters. One essential parameter is the topic entity from the graph. This entity serves as the starting point for the triple collection and is stored as the topic entity in the resulting `QAPair` object.
2. **Build Publication Subgraphs:** The first step is to collect the subgraph for each publication containing triples that conform to user-provided restrictions. This requires the *predicate type restriction* parameter, specifying the predicate identifier used to initially select relevant subgraphs. For each matching triple containing this predicate, the graph is traversed backward to locate the unique "paper type" triple that acts as the root node for the subgraph of the publication. Once the root is found, the strategy builds the subgraph by traversing outgoing edges until leaf nodes are reached. The result is a collection of publication subgraphs, each guaranteed to contain the specified predicate type.
3. **Extract Values:** In the second step, the strategy extracts values of interest from each publication subgraph found previously, using a *predicate value restriction* parameter defining the target predicate names. The strategy searches each subgraph for triples with matching predicates. Starting from these matching triples, it traverses all outgoing paths to leaf nodes, extracting their values. This produces a mapping from each extracted value of interest back to its source publication subgraph.
4. **Cluster Values:** These extracted values of interest are then embedded into a vector space using a selected embedding model and subsequently clustered using the DBSCAN [145] algorithm. This step forms clusters grouping semantically similar values. The mapping back to the source publication for each value is maintained throughout this process.
5. **Additional Restrictions:** Based on these semantically coherent clusters, additional restrictions, which are also provided as parameters, can be applied. Each cluster is processed, and relevant triples conforming to these new restrictions are extracted from the associated publication subgraphs and added to the data payload of the cluster. The

result is a set of clusters, each representing a semantic group of values and enriched with additional context triples.

6. **LLM-based Generation:** After the clustering is done, the next step is the generation of the question and answer. Here, a *template text* and *additional instructions* for the LLM are provided as parameters to the strategy. The template is a question with placeholders to guide the LLM in the generation process, and the additional instructions are appended to the base prompt to further fine-tune the generation process. To generate the question and answer, the clusters are processed one by one and forwarded to the LLM with the prepared prompt. The LLM then generates a question and a corresponding answer given the data from the cluster, the template and instructions.
7. **Prepare Ground Truth:** The triples accumulated within each cluster during the preceding steps serve as the ground truth. These are stored alongside the generated question and answer in the `QAPair` object.
8. **Manual Validation:** Due to the potential for LLM hallucination or failure to adhere to instructions or templates, a final manual validation step is essential. The generated question and answer are checked for correctness, coherence, and adherence to the template. Consequently, the relevance and correctness of the collected ground truth triples must be manually verified.

In the following, we illustrate an example to clarify the functionality of the clustering strategy. For this purpose, we use the following question template: "Which publications investigate the research object [research object name] and evaluate the sub-property [sub-property name]?".

First, we need to ensure that all subgraphs containing the required information are fetched for further processing. In the case of the ORKG, we can directly use the unique identifier of the *Research Object* predicate: *P*162024. This ensures that all publications that contain research objects are considered by the strategy.

Next, we define the list of predicate names that should be clustered from the subgraphs. Based on the question, we must specify the corresponding predicate names under which the requested information is stored in the graph. In our case, the predicates are labeled as `Research Object` and `Sub-Property`. In addition, further parameters can be defined to influence how the information is added to the clusters. For example, we can decide to split the clusters. In this case, the retrieved values are not added to the current cluster. Instead, a separate copy of the cluster is created for each value. This is desired for our question, as we do not want to collect all possible values of research objects and sub-properties in a single cluster but rather focus on specific instances.

At this step, each cluster contains all the publications of the dataset that investigate the same research object and evaluate the same sub-property. These clusters are now individually forwarded to the LLM that generates the question and the answer based on the triples in the cluster.

7.1.8.2. Subgraph Strategy

The second available strategy is the *Subgraph Strategy*, which generates diverse question-answer pairs related to a single publication at a time. Unlike the cluster-based approach, this strategy does not inherently generate questions spanning multiple publications. Furthermore, it is less restrictive, which can enable the generation of a wider variety of question types. The strategy operates as follows:

1. **Input Definition:** The strategy requires a publication entity from the KG as input, which acts as a *topic entity* identifier.
2. **Subgraph Extraction:** Starting from the publication entity, the graph is traversed to extract the subgraph of the publication by traversing the graph until the leaf nodes are reached. However, this subgraph is limited to a predefined size to fit within the context window of the LLM.
3. **LLM-based Generation:** This subgraph is then provided to the LLM with the instruction to generate both a relevant question and the corresponding golden answer. The generation process is guided by requiring the LLM to output a `JSON` structure containing the generated question, the answer and the specific subgraph triples that were used as the basis for generation.

7.1.8.3. Context and Answer Validation

During the generation of `QAPairs`, the LLM specifies golden triples that were used for the generation. We observed that this is not always accurate. Either the LLM indicates triples that do not match the generated answer, or the generated answer does not match the triples, indicating that the LLM is hallucinating. Therefore, we ensure through an additional validation process that the generated questions can actually be answered based on the data in the KG and that the created answer truly matches both the question and the data. The validation process works as follows:

1. **Triple Validation:** It may occur that initially, a larger set of triples was classified as relevant, but only a subset is actually necessary for answering the question. In this case, an LLM is prompted with a specially prepared validation prompt, and the triples as input. The LLM is instructed to ensure that all the information contained in the golden answer is present in the set of specified triples and to remove any triples that are not. This reduces the total set to the actually relevant triples or removes the question if no triples are relevant.
2. **Answer Validation:** Furthermore, another LLM call is made to verify whether the generated answer matches the question.
3. **Grammar Correction:** Since we observed that the generated questions are not always grammatically correct, an additional LLM call is performed to correct the generated question.

4. **Manual Validation:** Finally, the generated data is manually reviewed before being saved to the final `QADataset`.

7.1.9. App Component

The *App* component includes a Command Line Interface (CLI) application that enables users to operate the SQA system through the command line. It offers the following functionalities:

1. **Configuration Management:** All configurations of components in the SQA system can be generated using the command line. This ensures that the configurations are well-defined.
2. **Experiment Execution:** Experiments can be executed.
3. **Question Answering:** Run pipelines in interactive mode for question answering.

First, the CLI application enables configuration management. Since each configuration requires a different structure of the JSON file, it can be difficult for users unfamiliar with the system to create these files manually. Therefore, the CLI application provides guided configuration creation. This allows users to create configurations step by step and select the necessary parameters.

Furthermore, the CLI application facilitates direct execution of experiments based on created configurations. Users can select from a list of experiment configurations or create them directly. After executing the experiment, users receive a summary of results in the console, while detailed information is stored in a designated folder. This includes a CSV file containing all evaluations and experiment results. The results are also visually presented in diagrams that are saved in the folder. In addition, the configuration files are stored in JSON format in the folder to track which specific configuration led to the results.

Another feature of the CLI application is the interactive execution of pipelines. Users can select a pipeline configuration and ask a question. The pipeline is then executed for the question, and the answer is displayed in the console.

7.2. Implementation of HubLink

We implemented our proposed HubLink approach by closely following the pseudocode outlined in Chapter 4. For the implementation, we used object-oriented programming, distributing specific responsibilities across different classes. Moreover, to accelerate index building and partial answer generation, we implemented parallel execution using multiple threads.

For the vector store, we chose Chroma[3] because it can run locally without a server, making it easy to use and only requiring the installation of a Python package. HubLink is seamlessly integrated into the SQA framework and can be run using the configuration system provided. We implemented both proposed HubLink strategies to evaluate their performance on our KGQA dataset.

7.3. Accessing and Populating the Open Research Knowledge Graph

Our experiments are conducted on the ORKG as the underlying RKG. This section details the implementation and the setup of the ORKG for these experiments. We first outline the ORKG environment and the Application Programming Interface (API) used for the connection. Subsequently, we introduce the dataset of scientific publications employed in our study. We then describe the contribution templates designed to structure this data within the ORKG and the procedure for populating the graph accordingly. Finally, we address the measures implemented to ensure experimental repeatability and detail the methodology for selectively reading data from the graph that are relevant to specific experimental configurations from the ORKG.

7.3.1. ORKG Environment and API

The ORKG is hosted on three different environments. The *production*[4] environment provides access to the current version of the graph and is the most stable version. The *sandbox*[5] is a playground environment that is intended for experimentation on the ORKG. Finally, the *incubating*[6] environment is used to test new features that are still under development. We will conduct our experiments in the *sandbox* environment.

To access this environment, the backend is exposed over a RESTful API that is accessible online[7] and a Python package[8]. The RESTful API provides all read and write operations that are publicly available, and the Python package acts as a wrapper to provide easy access directly through code. However, at the time of writing, the package is still in development and does not yet provide the full list of features provided by the RESTful API. Consequently, we are using the Python package to store information in the graph, and we use the RESTful API for reading operations.

[3] `https://www.trychroma.com/` [last accessed on 25.04.2025]
[4] `https://orkg.org/`
[5] `https://sandbox.orkg.org/`
[6] `https://incubating.orkg.org/`
[7] `https://tibhannover.gitlab.io/orkg/orkg-backend/api-doc/`[last accessed on 09.04.2025]
[8] `https://orkg.readthedocs.io/en/latest/index.html`[last accessed on 09.04.2025]

Data Item	Description	Type
Paper Class	A general classification of the publication.	Meta Data
Research Level	Distinguishes on whether the research is collected firsthand.	Meta Data
Kind	Classifies whether the paper can be seen as a full research paper.	Meta Data
Research Object	The investigated object(s) of research.	Content Data
Tool Support	Indicates whether the paper employed a tool.	Content Data
Input Data	Indicates whether the paper used specific input data.	Content Data
Replication Package	Indicates whether the paper provides a dedicated replication package.	Content Data
Threats to Validity (TtV)	The threads to validity that are named in the paper.	Content Data
TtV Guideline	Indicates whether the paper references TtV guidelines.	Content Data
Evaluation Method (EM)	The applied evaluation method.	Content Data
EM Guidelines	Indicates whether the paper referenced guidelines for EM.	Content Data
Property	The property that is evaluated with a EM for a research object.	Content Data

Table 7.1.: Extraction data schema applied in the work of Konersmann et al. [141]

7.3.2. Dataset of Scientific Papers

To evaluate how well HubLink performs in the context of scholarly literature searches, we require a dataset of scientific publications. For this purpose, we use the dataset provided by Konersmann et al. [141]. This dataset was originally created to analyze how SWA research objects are evaluated and how replication packages are provided. It was created through a literature search and annotations were extracted according to a specific schema. In their study, a total of 153 publications were included according to the following inclusion and exclusion criteria:

1. Papers presented at European Conference on Software Architecture (ECSA) and International Conference on Software Architecture (ICSA) conferences between 2017 and 2021.
2. Comprehensive technical papers, excluding short papers, experience reports, and opinion pieces.

3. Papers focusing on evaluation research, validation studies, solution proposals, and philosophical discussions.

The schema employed for this annotation process is presented in Table 7.1. Each publication is annotated according to its research objects, research level, paper class, and validity information. The table also categorizes data as either metadata or content data. The differentiation between metadata and content data presented in Table 7.1 is based on the information provided by Konersmann et al. [141].

According to Riley [146], metadata is defined as "data that provides information about other data". In the context of the data schema defined in Table 7.1, two types of metadata are present: *descriptive* and *preservation*. Descriptive metadata provides information for finding or understanding a resource, such as the title, authors, and publication year. Preservation metadata, a subtype of administrative metadata, encompasses information regarding the long-term management of a file [146].

In addition to metadata, the dataset also contains content data, which we define as information *that is contained within* the scientific artifact. This implies that to obtain the desired content data, the text of the publication must be read.

The extracted annotations in the dataset are based on the content of the publications and have been scientifically validated. Combined with the corresponding metadata, this dataset is well suited for our experiments, as it allows us to formulate questions regarding both metadata and content. We therefore compiled the data and consolidated them into a single JSON file to be used during our experiments. The resulting dataset consists of 153 publications, with the annotations shown in Table 7.1. In addition to the annotations, the dataset also includes metadata such as title, authors, DOI, and publication year.

7.3.3. Contribution Templates

To add new scholarly contributions to the ORKG, the graph allows users to include publications by providing a DOI or the title. The publication is then either linked to an existing resource in the graph or a new resource is created. In both cases, the system automatically fills in the metadata. After this step, the graph contains only the metadata of the paper. To specify what scientific contributions the paper makes, this information is added using the *Contribution* ORKG content type. In other words, all the knowledge presented in a paper is added to the graph through one or more contributions [51, pp. 58–60].

For the purpose of our experiments described in Section 7.3.2, we needed to load the labels provided by [141] into the ORKG. To do so, we created templates for individual contributions to attach to each publication. We decided to design four different graph variants in order to test the robustness of the HubLink retriever against these variations. These variants are shown in Figure 7.9. The idea behind splitting the content into different variants is to maintain the same informational content while varying the depth and breadth at which it is stored. Depth refers to how far the information is stored from the root node of the paper, whereas breadth refers to whether the information is semantically separated across

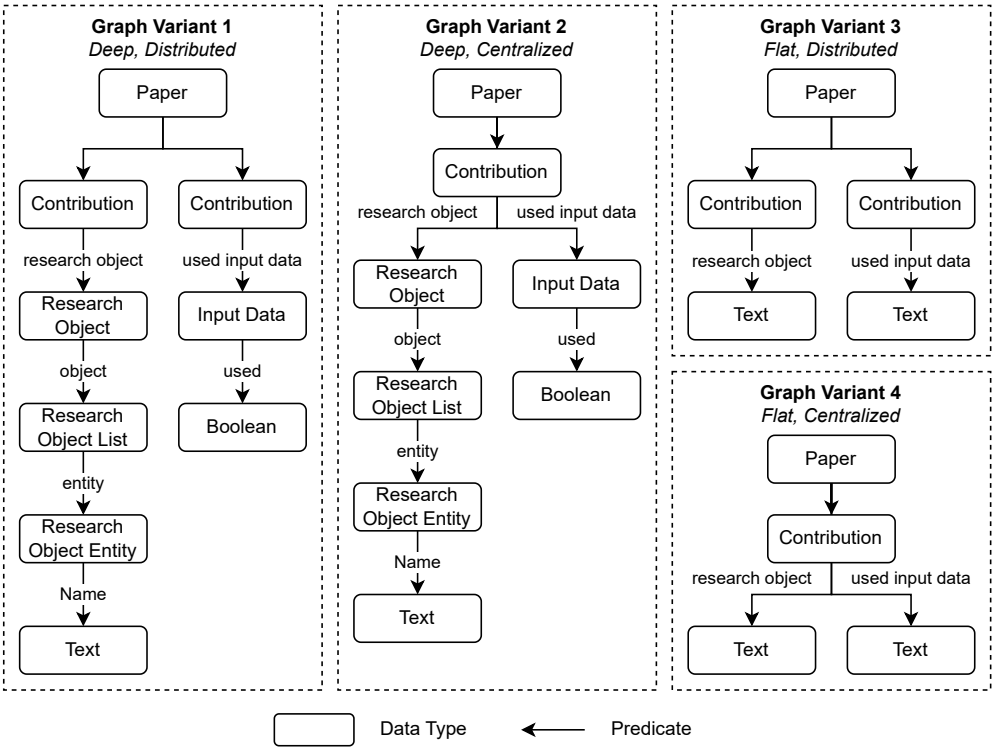


Figure 7.9.: Excerpt of the four different graph variants of the ORKG featuring the fields research object and input data. This figure highlights the difference between the depth and width of how the same information is stored in each of the different graph variants.

multiple contributions or centrally stored within a single contribution. The following four variants have been prepared:

GV1 This graph variant stores data in long paths. In addition, the information is semantically separated and distributed between different contributions.

GV2 This graph variant stores data in long paths. All information is collected in a single contribution.

GV3 This graph variant stores data in short paths. The information is semantically separated and distributed across different contributions.

GV4 This graph variant stores data in short paths. All information is collected within a single contribution.

An illustrative example depicting an excerpt from each of the variants is presented in Figure 7.9, while the full templates are provided in Appendix A.2. The figure shows the root entity of a paper and how information about input data and research objects is stored for each of the four different variants. This highlights the difference in depth and breadth in which the same information is stored between templates.

Using these variants, our aim is to test how robust HubLink is with respect to different graph structures. Long paths allow more information to be stored, but also require more processing. In contrast, short paths limit the amount of stored information, which may result in the loss of semantic context. In addition, we want to examine whether the distribution of data between one or multiple contributions has an effect on performance. Moreover, templates differ in the types that they use to store the data. Consequently, if a KGQA approach can be applied to any of the graph variants without requiring a change in its configuration or implementation, it indicates that the approach is *schema-agnostic*.

The most difficult design decision during the creation of the templates was the multiple classification types in our data. These involve the classification of a paper according to a specific term from a specific selection of terms. For example, whether the paper class is evaluation research, a proposal for a solution, or validation research. To store such information, a typical user interface would provide a *multiple-choice* data type. However, the template system provided by ORKG does not provide such a datatype at the time of writing. We therefore needed to create such a multiple-choice concept ourselves with the structures provided by the ORKG. This encompasses several considerations that must be taken into account.

We considered two choices to realize multiple choice. Using *plain text* would allow users to flexibly input their choices. Nevertheless, this approach would not enable the template author to constrain the input, which would be against the idea of consistency in the data, because contribution data types that may relate to the same concept are not guaranteed to have the same value. This includes simple issues like casing, typos, synonyms, etc. Another realization would be a *List of Boolean values*. This would allow the template author to create a list with predefined choices that are relevant. Each choice would be of the type Boolean to determine whether the choice is selected or not. However, the drawback of this approach is that it is less flexible for future extensibility. If the list is later updated to include more choices, all papers that have already used the template will have an empty value for the new choice. However, solving this issue is beyond the scope of this thesis. We implemented both methods in our templates. The variants **GV1** and **GV2** implement the list of Boolean, while the variants **GV3** and **GV4** store each choice in plain text. The concrete templates for each variant are provided in the Appendix A.2.

7.3.4. Populating the ORKG with Data

To store data in the ORKG graph using various templates, a dedicated script was implemented. This script reads the metadata and annotations of the publications from a JSON file and processes each entry iteratively. For each publication, it checks whether it already exists in the graph with all specified contributions. If not, the corresponding contribution is added and populated with the relevant data. The script is triggered automatically whenever the SQA system initializes the ORKG class, which happens before each experiment is run.

7.3.5. Ensuring Repeatability

During the implementation of the ORKG, it became evident that inserting data into the graph leads to the automatic generation of unique resource IDs. These IDs cannot be manually controlled and will change if data is deleted and then reinserted into the graph. This poses a challenge since the KGQA dataset stores these resource IDs as part of the ground truth to verify whether the correct triples are retrieved during evaluation. If the original data are removed and later reinserted, new resource IDs are generated, which invalidate the KGQA datasets.

To address this issue, a script was developed that compares the resource IDs stored in the KGQA dataset with the IDs in the graph and updates the IDs in the dataset accordingly. With this approach, we can ensure that the experiments remain reproducible over time. Specifically, this means that the publications can be automatically reinserted into the ORKG using the SQA system in the case that they have been deleted. The system then runs a script that updates the corresponding IDs in the KGQA dataset.

7.3.6. Reading from the ORKG

To retrieve data stored in the ORKG, we use the provided RESTful API. This API was integrated into the SQA system to enable the automated execution of the experiments. However, during implementation, a key challenge emerged. The ORKG contains a large amount of data not relevant to our experiments. These additional data introduce noise, making it difficult to establish a controlled experimental setting. Another issue we encountered is that standard users have limited ability to delete data in the ORKG. This became an issue with our four different graph variants, as the individual deletion of contributions was not possible. Consequently, we stored all variants simultaneously in the graph.

To ensure that an experiment operates only on the data relevant to the experiment, a tailored solution was implemented. During the initialization of the ORKG, a preprocessing step generates a locally stored list of valid entities and triples. This involves examining all entities and triples in the research field *Software Architecture and Design*, which contains the data relevant to our experiments. Each entity and triple is checked to determine whether they are part of the metadata of a relevant publication or belong to a contribution associated with the currently selected graph variant. Only those triples and entities that meet these criteria are included in the list.

At runtime, this list acts as a filtering mechanism: for each read request made by a retriever, only those triples and entities contained in the prepared list are returned. This ensures that only the intended data for the specific experiment, and exclusively from the selected graph variant, are taken into account.

7.4. Baseline KGQA Retrieval Approaches

To evaluate our HubLink approach against existing KGQA approaches, we implemented five distinct baseline methods within the SQA system. This section details the selection process and the implementation of these baselines. First, we outline how candidate approaches were identified and filtered. Then, we explain the methodology of each selected retriever and briefly describe its implementation.

7.4.1. Selection of Baselines Approaches

To compare HubLink with state-of-the-art methods from the literature, we selected and implemented several approaches drawn from recent publications. The chosen baseline approaches represent established methods within the field previously evaluated on open-domain KGQA benchmarks. In the following, we outline the systematic process through which these methods were chosen.

7.4.1.1. Collecting Paper Candidates

Recently, surveys have been published that structure the current approaches found in the literature. Pan et al. [14] provide an in-depth analysis of the integration of LLMs with KGs, which includes KGQA but also goes beyond that. In their work, they propose a categorization of different integration strategies, assigning each examined paper to one of these categories. From this structure, we selected the categories relevant to KGQA, namely *KG-Enhanced LLM - Inference*, *LLM-Augmented KG - Question Answering*, and *Synergized Reasoning*, as these directly address the integration of LLMs and KGs for question-answering tasks.

Another survey by Peng et al. [13] proposes a taxonomy for GraphRAG approaches, which classifies the methods in a range of dimensions. From this set, we include all publications covered by the survey, except those classified as *Non-parametric*, *GNN-based* retrievers, or those considered *Training-based*.

In addition to the surveys, we conducted a Google Scholar search to identify further KGQA approaches. Since both surveys were published in 2024, we limited our search to this year in order to find additional approaches not yet captured by the surveys.

7.4.1.2. Excluding Papers not Relevant

Through the above-mentioned surveys and Google Scholar search, we collected an initial pool of 76 publications. The next step was to identify the KGQA approaches most relevant for comparison with HubLink by applying the following exclusion criteria to each of the publications:

[C1] LLM-Based: The approach proposed in the publication must employ a pre-trained LLM to support the retrieval process. Embedding models are also included under this criterion. This is relevant because our objective is to explore how LLMs can support literature search within a QA context.

[C2] KGQA Approach: The approach proposed in the publication must represent a generalizable KGQA approach. Specifically, it should accept a question in natural language and a KG as input, with the goal of extracting relevant information from the KG to answer the question.

[C3] Training-Free: The approach proposed in the publication must not require additional training or fine-tuning of pre-trained LLMs, nor the training of other models such as Graph Neural Networks (GNNs). Consequently, all approaches that depend on a dataset of training examples have been excluded, as we lack the resources for extensive training.

Applying these criteria to the collection of 76 publications resulted in 13 relevant papers. Specifically, one paper was excluded for not using an LLM (**C1**), 21 were excluded as they did not represent a suitable KGQA approach (**C2**), and 41 were excluded for requiring model training (**C3**). The complete list of candidates and the filtering results are provided in the replication package [124].

7.4.1.3. Assessing Implementation Feasibility

For the remaining 13 papers deemed relevant, we evaluated the availability and applicability of their implementations provided by the authors for integration into the SQA system.

The approaches RoK [98] and KSL [89] were excluded as they do not provide source code. Their complexity made reimplementation impractical without access to the original code.

In the case of KG-GPT [97], after reviewing the code repository[9], we found that the implementation corresponds to a claim verification pipeline rather than a traditional QA setting. This assumes a prior mapping of claim entities to graph entities, which is not available in our use case.

For ODA [90], although the authors provide a repository[10], it contains only graph and dataset resources, lacking the implementation of the ODA approach itself.

DiFaR [99] does not have a public implementation. However, its methodology closely resembles the RAG framework [8], differing mainly in embedding graph triples instead of documents. Given the experience of the thesis author with similar architectures, we deemed reimplementation feasible based on the description of the paper.

For the remaining methods: StructGPT [92], ToG [88], Mindmap [96], ToG-2 [94], GoG [95], GRAPH-COT [91], and FiDeLiS [93], we found that the provided source code was generally adaptable for integration into the SQA system.

[9] `https://github.com/jiho283/KG-GPT` [last accessed 24.11.2024]
[10] `https://github.com/Akirato/LLM-KG-Reasoning` [last accessed 24.11.2024]

7.4.1.4. Deciding on Final Implementations

After assessing implementation feasibility, eight of the 13 methods remained as candidates: StructGPT, ToG, Mindmap, ToG-2, GoG, GRAPH-COT, FiDeLiS, and DiFaR. To keep the scope of this work manageable, we ultimately selected five of these for implementation, guided by their methodological diversity. We categorize the eight candidates as follows:

- **Stepwise reasoning:** These approaches iteratively query the LLM to derive an answer step by step. This category includes: StructGPT [92], ToG [88], ToG-2 [94], GoG [95], GRAPH-COT [91], and FiDeLiS [93].
- **Subgraph construction:** These methods focus on building relevant subgraphs from which information is extracted. Mindmap [96] is the only candidate in this category.
- **Embedding-based:** These methods primarily use dense vector representations for retrieval. DiFaR [99] is the only candidate here.

During the conceptual phase of this work, prior to this baseline selection, StructGPT [92] and ToG [88] were implemented to evaluate the general feasibility of the thesis. At the time, both were highly cited and provided an adaptable public code. However, these approaches are very similar and share a weakness particularly relevant to scholarly literature search, as their entity selection can become random beyond a certain threshold, making correct entity identification dependent on chance. This significantly impacts the quality of the answer, as detailed in Section 10.3.

Although ToG-2 [94], a successor to ToG, was published during the conduct of this thesis, the issue described above was not resolved in the new version. For this reason, we decided not to implement this approach. Instead, we selected FiDeLiS [93] from the Stepwise reasoning category, as it specifically addresses the entity selection problem using an embedding-based similarity assessment. This brings the total number of implemented methods in the Stepwise reasoning category to three.

In the Subgraph construction and Embedding-based categories, only Mindmap [96] and DiFaR [99] remained after filtering. Therefore, both were implemented as baselines, bringing the total number of baseline methods implemented to five.

7.4.2. Direct Fact Retrieval (DiFaR)

Direct Fact Retrieval (DiFaR) is a KGQA retrieval approach proposed by Baek et al. [99]. It was evaluated on fact retrieval tasks across two different domains: QA and dialogue generation. For QA, three different datasets were used: SimpleQuestions and WebQuestionsSP for the Freebase graph, and Mintaka for the Wikidata graph. For dialogue generation, they used the OpenDialKG dataset designed for the Freebase graph. Their tests show that the approach outperforms all baselines, although performance is lower for intrinsically more complex multi-hop retrieval questions.

7.4.2.1. Approach Explanation

The retriever first undergoes an indexing phase, during which it converts all triples in the KG into a set of embeddings. At query time, the question is embedded using the same embedding model that was used for the triple conversion. Subsequently, a nearest-neighbor search is performed to find the triples whose embeddings are closest to the question embedding, thus enabling a quick search through potentially billions of dense vectors. These triples serve as the context for answering the question. The researchers further propose a refinement, termed DiFaR2, involving a reranking of the retrieved triples. This approach utilizes a language model provided with both the question and the retrieved triples to evaluate triple relevance.

7.4.2.2. Approach Implementation

We implemented the DiFaR approach based on the descriptions provided in the paper. In our implementation, the indexing process starts by selecting an initial entity within the graph and then traversing sequentially from it. Each triple collected during traversal is then embedded using a pre-trained embedding model. The specific embedding model is selected based on the provided configuration file. These vectors are then stored in a vector store. At query time, the question is processed to generate its embedding. This embedding is used in a ANN search on the data stored in the vector store. The triples retrieved from this search are then incorporated into an LLM prompt for the generation of the final answer. Regarding the proposed reranking, the SQA system already provides a post-retrieval procedure that reranks contexts based on relevance, which can be enabled via configuration.

7.4.3. Think-on-Graph (ToG)

Sun et al. [88] propose ToG, a KGQA retrieval approach based on beam search. The authors evaluated their approach on nine different datasets for the Freebase and Wikidata graphs to demonstrate its advantage in reasoning over knowledge-intensive tasks. The datasets used for Freebase were ComplexWebQuestions, WebQuestionsSP, GrailQA, SimpleQuestions, and WebQuestions. For the Wikidata graph, the datasets were QALD10-en, T-REx, Zero-Shot RE, and Creak. Their tests show that ToG achieves state-of-the-art performance on six out of the nine datasets.

7.4.3.1. Approach Explanation

The process is initialized with topic entities, which act as entry points into the graph. Starting from these entities, an exploration takes place. During exploration, the system iteratively traverses the KG to build reasoning paths. At the start of each iteration, the current set of reasoning paths includes all entities and relations discovered so far. The LLM identifies candidate relations by querying the KG for relations connected to the tail entities

of the paths of the previous iteration. These relations are ranked by their relevance to the question. The top N are then selected using an LLM-based pruning step to narrow the search space. Next, the LLM uses the selected relations to find candidate entities, which are then randomly pruned to stay within a predefined threshold specified in the parameters. The reasoning paths are updated with the newly discovered entities and relations, effectively increasing their depth by one with each iteration.

The reasoning phase follows, involving evaluation and potential termination. The LLM evaluates whether the current reasoning paths contain enough information to answer the question. If so, it generates an answer using these paths. If not, exploration continues until either an answer can be formulated or a predefined maximum depth is reached. If sufficient information is not found by then, the LLM resorts to its internal knowledge to produce a response.

7.4.3.2. Approach Implementation

The original code provided by the authors is available online[11]. We adapted their code with minimal necessary changes to work with the SQA system interface. During testing, we encountered several issues requiring further adjustments. First, many executions failed because some outputs of the LLM deviated from the expected output format. We found that the original parser was unable to handle these variations in LLM output. To address this, we developed a more robust parser capable of extracting information from a wider range of LLM outputs. Second, we parallelized the entity searching and scoring processes, as the original sequential implementation was inefficient. Third, the original implementation only returned the LLM-generated answer. We extended the output to also return the triples used to generate the answer to allow for evaluation of retrieval performance. Finally, many of the prompts used by the retriever are few-shot prompts requiring examples. In the original code, these prompts targeted a different knowledge base, so we modified the examples to align with our label-based QA dataset.

7.4.4. StructGPT

Another KGQA retrieval approach based on beam search is StructGPT, proposed by Jiang et al. [92]. In their work, the authors explore reasoning over multiple types of structured data, including tables, KGs, and databases, within a unified paradigm. Consequently, they evaluated StructGPT on a wide range of tasks, including KGQA, table-based QA, and text-to-SQL, using a total of 9 different datasets. For KGQA, they tested WebQuestionsSP and MetaQA. For table-based QA, they used TabFact, WikiTableQuestions, and Fever. For text-to-SQL, the datasets were Spider, WikiSQL, and SParC. The authors claim that StructGPT enhances the reasoning performance of LLMs on structured data in zero-shot and few-shot settings, achieving results comparable to competitive, fully supervised methods.

[11] `https://github.com/IDEA-FinAI/ToG/` [last accessed 21.11.2024]

7.4.4.1. Approach Explanation

The retrieval process begins with a designated topic entity, serving as the entry point into the KG. The method first aggregates all unique relations associated with the topic entity. These relations then undergo preprocessing steps to filter out redundancies and to linearize the remaining relations into a simple string format suitable for LLM input. The LLM is responsible for selecting a single relation per iteration to guide the traversal path. In the first iteration, it selects one relation deemed relevant to the query. In subsequent iterations, while considering the history of previously selected relations, the selection process still yields only one new relation.

Once a relation is selected, all entities connected to the current entity via the selected relation are gathered from the graph. These retrieved triples are subsequently classified based on their type. However, we observe that this classification appears relevant only for the Freebase graph, which we do not use in our project. Following the classification, the LLM examines the retrieved triples to determine whether the information is sufficient to answer the query. During this verification, the number of triples considered is constrained to a predetermined maximum to limit the context size that is queried to the LLM. If the LLM deems the information adequate, it generates the answer. Otherwise, the procedure continues with the next iteration, with the LLM selecting another relation and the retrieval of additional triples. This process continues until an answer is generated or the maximum number of iterations is reached.

7.4.4.2. Approach Implementation

The implementation of StructGPT is publicly available online[12]. We adapted the original code with minimal modifications necessary for compatibility with the SQA system interface. During implementation, we observed that the traversal mechanism did not operate as intended in the original code because the main loop terminated prematurely using an unconditional break statement after the first iteration. Because the descriptions in the paper and the surrounding code logic suggests iterative traversal, we removed this break statement, allowing the loop to execute up to the specified maximum iterations.

Additional modifications were necessary. We observed excessive runtime and consequently implemented parallelization for retrieving and processing relations for each entity. Furthermore, the original implementation could not traverse the edges of entities against the direction of the graph, which impairs the ability of the retriever to provide answers to many questions in the KGQA datasets used in our experiments. We addressed this by adding the functionality for bidirectional retrieval. Lastly, the original implementation returned only the generated answer. We extended the output to also include the retrieved triples to be able to assess the retrieval part of the approach. For this to work, we needed to add an LLM-based filtering of the triples, as the total number of triples from which the answer is generated is very large.

[12] `https://github.com/RUCAIBox/StructGPT/` [last accessed 21.11.2024]

7.4.5. FiDeLiS

The FiDeLiS retriever is another beam search-based method proposed by Sui et al. [93]. The approach was evaluated on the Freebase and Wikidata graphs. For Freebase, the datasets WebQuestionsSP and ComplexWebQuestions have been tested. For Wikidata, the CR-LT-KGQA dataset was used. Their tests show that the approach outperforms existing baselines across all datasets.

7.4.5.1. Approach Explanation

The approach is initiated with a question and a corresponding topic entity, which serves as the entry point in the graph. Then, an LLM generates a strategic plan to address the question, including extracting relevant keywords and converting the query into a declarative statement. Subsequently, the extracted keywords are embedded using a pre-trained embedding model. The main iterative process then begins by retrieving all relational paths associated with the current entities, starting from the topic entity. Both the predicates and their associated tail entities are embedded and subsequently scored based on similarity to the keyword embeddings. The resulting relations are ranked, and only the top N are retained and added to a cumulative list of candidate paths. If the maximum path length has not been reached, the top-K candidates from this list are selected for further expansion in the next iteration, guided by the previously generated plan. At each iteration, a deductive termination check determines whether the process should halt, and a final answer can be synthesized from the candidate paths. The loop continues until the step limit is reached or an answer is produced.

7.4.5.2. Approach Implementation

The implementation of FiDeLis is publicly available online[13]. We adapted the original code with minimal necessary changes to work with the SQA system interface. However, some modifications were required. First, we had to adapt the output parsers to be more robust when working with various LLMs, as the original parsers required the LLM output to match the expected format exactly, which is not always the case in reality. Second, we adapted the code to return the retrieved triples, enabling evaluation of the retrieval performance of the method. Third, we encountered long execution times. To accelerate retrieval, we implemented a caching mechanism to avoid redundant graph queries and parallelized the entity scoring process.

7.4.6. Mindmap

The Mindmap retriever, proposed by Wen, Wang, and Sun [96], is a KGQA retrieval approach that extracts relevant entities from a question and constructs evidence subgraphs. The

[13] `https://anonymous.4open.science/r/FiDELIS-E7FC` [last accessed on 03.02.2025]

approach was evaluated in the medical QA domain using three datasets: GenMedGPT-5k, GMCQA, and ExplainCPE, covering scenarios such as patient-doctor dialogues, clinical dialogues, and multiple choice questions from a pharmacist examination. They constructed two KGs that contain medical concepts and relationships, demonstrating state-of-the-art performance.

7.4.6.1. Approach Explanation

The retriever starts with a preprocessing step where all entities within the KG are embedded. At query time, it prompts an LLM to extract entities from the input question. Based on these extracted entities, a nearest neighbor search identifies the entities most semantically similar within the KG. For each identified graph entity, the retriever constructs two types of evidence subgraphs: (1) the shortest paths connecting the identified entities within the graph and (2) the 1-hop neighbors of each identified entity. Subsequently, the content of both subgraphs is transformed into natural language descriptions using an LLM. The final answer is generated by prompting the LLM with the natural language descriptions and the original question, allowing the model to synthesize information into a coherent response.

7.4.6.2. Approach Implementation

The implementation of the Mindmap retriever is publicly available online[14]. We adapted their code with minimal necessary changes to integrate with the SQA interface. However, several more substantial modifications were required for the retriever to function correctly in our setup.

First, the retriever requires a shortest path algorithm to function. The original implementation relied on the built-in shortest-path functionality provided in the graph framework. Since the SQA system is designed for generic RDF graphs lacking this specific feature, we implemented a bidirectional breadth-first search algorithm to find the shortest paths between entities in the graph. Second, the original code relied on precomputed text files for entity embeddings. To handle arbitrary graphs and questions dynamically, we replaced this with an on-the-fly embedding process, storing entity vectors in a vector store and computing question embeddings during retrieval. Third, we adapted the prompts, correcting grammatical errors (possibly due to translation introduced by the original authors) and modifying the few-shot examples for our label-based KGQA dataset. Fourth, we parallelized the queries that retrieve all neighbors of an entity because we identified this process to be a performance bottleneck. Finally, we ensured that the retriever also returns the triples found during retrieval, which allows the evaluation of the retrieval component of the approach.

[14] `https://github.com/wyl-willing/MindMap/tree/main`[last accessed on 26.02.2025]

8. Evaluation Preliminaries

This chapter prepares the evaluation as well as the parameter selection process. In the following, in Section 8.1, we first go into detail about the evaluation concept which we used to perform the evaluation. Then, in Section 8.2, we describe the environment in which the experiments and the parameter selection process have been conducted. Next, Section 8.3 documents how the KGQA datasets used in this thesis were created. Following this, Section 8.4 explains the evaluation framework, introducing the goals, concepts, and metrics for the evaluation. Finally, Section 8.5 describes the dependent and independent variables of the experimentation.

8.1. Evaluation Concept

This section details the evaluation concept, which we used to perform the evaluation with the objective of assessing the performance of our proposed HubLink approach. The primary objective of this concept is to systematically compare the capabilities of the HubLink retriever against established baseline methods. As indicated in Section 7.3, the ORKG serves as the underlying RKG for these experiments. Furthermore, the KARAGEN method [53], designed for implementing an LLM-based QA system on the ORKG, forms the basis of the experimental setup. This framework is particularly suitable, as it encompasses the necessary components for applying a KGQA retriever to the ORKG, including data population and retrieval processes. Figure 8.1 provides a visual overview of the complete experimental design, which we detail in the following.

The general experimental concept starts with the *Knowledge Graph Generation and Population* module. This module is proposed by the KARAGEN method and includes the preparation of contribution templates to populate the ORKG with data. We prepared four such templates, which are described in Section 7.3.3. We fill these templates with data using the SWA schema detailed in Section 7.3. These populated templates are used by the SQA system to automatically fill in the publication data in the ORKG.

The next module in the evaluation concept is *Knowledge Retrieval*, which also aligns with the principles described in the KARAGEN method. In our implementation, the SQA system is employed to construct the KGQA retrieval pipeline. This enables systematic experimentation by providing a dynamic configuration setup where parameters and pipeline steps can be easily exchanged. The pipeline accepts a natural language question as input and, optionally, a topic entity serving as an entry point into the ORKG. It then performs knowledge retrieval utilizing either the HubLink retriever or one of five baseline KGQA approaches from the

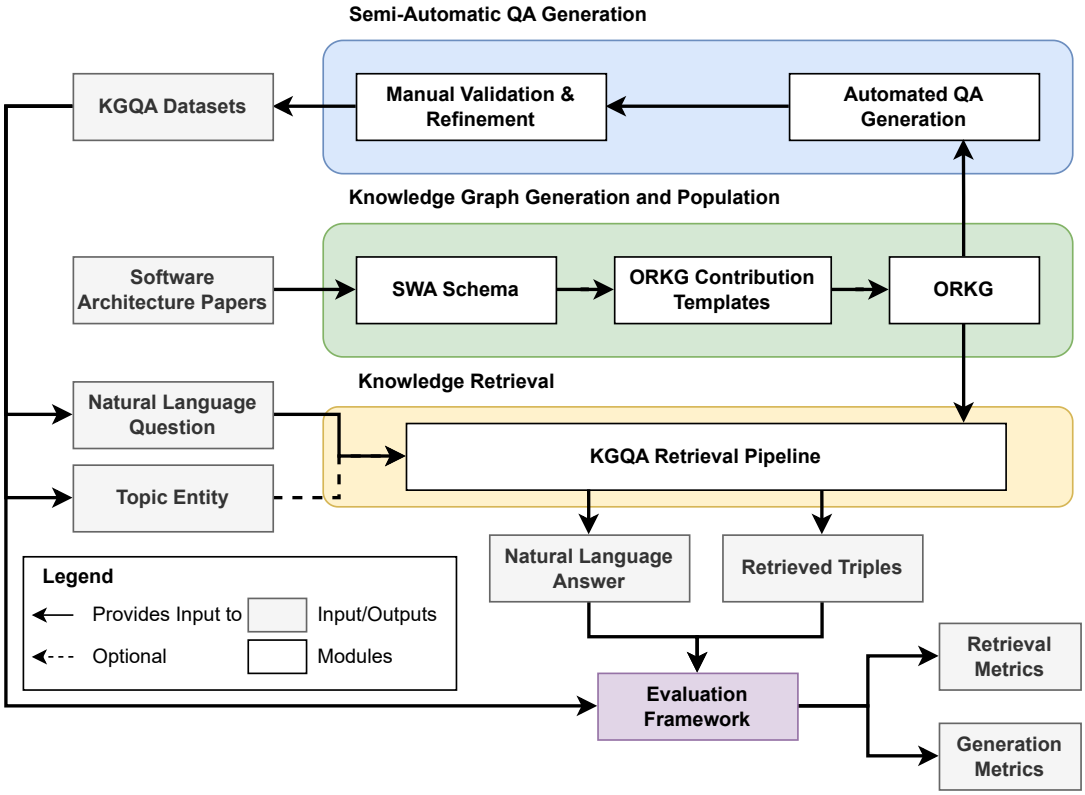


Figure 8.1.: The overall evaluation concept that we used to perform the experiments on the ORKG using the proposed HubLink approach and the implemented KGQA baseline approaches.

literature, detailed in Section 7.4. The parameters that are used for these approaches are chosen by the *Parameter Selection Process*, which is documented in Chapter 9. The outputs of the *Knowledge Retrieval* module are a natural language answer and the list of triples retrieved from the graph. These two types of outputs allow the evaluation of both the generation and the retrieval parts of the KGQA approach.

The inputs to this system (question and topic entity) are automatically provided to the KGQA approach by the KGQA dataset. This dataset is the output of the *Semi-Automatic QA Generation* module. Here, the QA generation processes implemented in the SQA system are used for the generation of KGQA pairs (see Section 8.3). During generation, questions are generated in a controlled manner with LLM assistance using the ORKG subgraph as the data source. Furthermore, ground-truth data is generated, which is required for the subsequent evaluation phase.

Following the execution of the retrieval pipeline, the output is evaluated within the *Evaluation Framework* module. This final module employs the generated ground-truth data to compute relevant performance metrics. Analysis of these metrics facilitates drawing conclusions regarding the effectiveness and performance characteristics of the evaluated KGQA approaches.

8.2. Software and Hardware Environment

In this section, we detail the environment in which the experiments for this thesis were executed.

The experiments were performed on the ORKG in the sandbox environment as detailed in Section 7.3. However, this does not affect the generality of the results, as the ORKG serves as a representative example of an RKG [12]. Consequently, the broader relevance of the obtained results are not compromised.

Moreover, all experiments were executed within the SQA framework, which was developed by the author of the thesis and is detailed in Section 7.1. Both the proposed HubLink retriever and the baseline retrievers were integrated into this framework. To ensure reproducibility, each experiment configuration is specified in a dedicated file using the JSON format. This facilitates straightforward replication of the experimental setup.

With respect to hardware, all experiments were performed in the same consistent hardware and software environment. The system operates on the Linux kernel version *6.1.0-23-amd64*, equipped with a *Intel(R) Xeon(R) Gold 6258R CPU @ 2.70GHz* processor and a *Tesla V100S-PCIE-32GB* graphics processing unit. The implementation relies on a *Python 3.12* environment, with a comprehensive list of software packages and their versions provided in the accompanying replication package [124].

Furthermore, the experimental setup uses both local and remote LLMs. Local LLMs are managed and executed via the Ollama framework[1]. Remote access LLMs is facilitated through the OpenAI API[2].

8.3. Creating the KGQA Datasets

To be able to evaluate the retrieval capabilities on diverse retrieval operations and specific use cases in scholarly literature searches, we created new KGQA datasets (**C3**). These datasets have been created with the help of the KGQA retrieval taxonomy (**C3**) and six use cases using the semi-automatic generation procedures provided by the SQA system (see Section 7.1.8).

In the following, we first introduce the use cases for scholarly literature search. Then, we explain the different levels of content granularity to clarify the level at which the new datasets are situated. Finally, we introduce the new KGQA datasets for four different graph variants.

[1] `https://ollama.com/` [last accessed on 24.02.2025]
[2] `https://platform.openai.com/docs/overview` [last accessed on 24.02.2025]

8.3.1. Use Cases for Scholarly Literature Search

To create the KGQA datasets, we prepared use cases. These use cases serve to support the creation process by incorporating real-world scenarios. To guide the development of use cases, we use the categories *Answer Type* and *Condition Type* from the KGQA retrieval taxonomy introduced in Chapter 6. Here, *Condition Type* refers to the types of conditions provided with a question. The *Answer Type*, on the other hand, implicitly classifies the types of information expected in the answer.

For the design of the use cases, we decided to adapt these two categories to make them better suited to our experimental data. This decision was based on the fact that both categories originally included a large number of classes, which would result in an unmanageable number of use cases if all combinations were considered. Consequently, because the labeled data that we use consists of two types of data (metadata and content data), as shown in Table 7.1, we apply this distinction by adapting the answer and condition types for the creation of use cases. The result is the following adaptation of the two categories:

Adapted Answer Type Classifies whether a question expects a *metadata* or *content data* type in the answer.

Adapted Condition Type Classifies whether a question contains a condition of the *metadata* or *content data* type.

Based on these adapted categories, we present six different use cases tailored to the scholarly literature search task.

Use Case 1

The first use case reflects the current state of practice in scientific literature search. The researcher seeks additional details about the *metadata* of one or more papers. To find this information, the researcher provides the KGQA system with specific metadata information related to the papers they are interested in. In response, the KGQA system returns information on other *metadata* attributes of the papers rather than content information.

The first use case is classified as *Adapted Condition Type: Metadata* because all questions related to this use case require the answer to meet a specific metadata condition. Furthermore, this use case is categorized as *Adapted Answer Type: Metadata*, as only metadata information is expected in the response.

A practical example of this use case would be a researcher asking for the publications of a specific author. In this case, the name of the author serves as the input constraint, while the returned publication titles represent the output type. Both the input and output consist solely of metadata. Similarly, another example would be a question that provides keywords. The KGQA system then has to return the titles of the papers that are related to those keywords.

Consequently, this use case reflects the current way of searching for scientific artifacts, since state-of-practice search engines like Google Scholar are expected to receive keywords and return titles of publications.

Use Case 2

In the second use case, the researcher seeks information about the *content* of one or more papers. In this use case, the researcher provides the KGQA system with *metadata* information about the papers and asks a question about their contents. The KGQA system is then expected to extract content information related to specific papers that conform to the metadata constraints provided.

The second use case is classified as *Adapted Condition Type: Metadata*. However, its category regarding the second dimension is *Adapted Answer Type: Content*, as the answer is expected to contain content information. From a retrieval perspective, we expect this use case to be more challenging, as it requires the KGQA system to search through the extensive content information provided in the papers to find the expected answer.

A practical example of this use case would be a researcher asking for the conclusions that a specific publication has drawn. In this case, the name of the publication serves as the input constraint, where the KGQA system is required to search the content of the specific publication. The expected output would be the conclusions drawn in the publication, which is content-based information as it requires the KGQA system to reason over the content of the publication. Another scenario that would fit this use case would be a researcher asking the KGQA system what research problems an author has worked on. In this case, it would require retrieving information from all publications of the author, extracting and aggregating the research problems that each publication investigated, and returning the aggregated list of research problems as the answer.

This use case shows the potential for KGQA to help vastly improve the literature search process. By directly allowing researchers to find information based on the contents of a paper, the time and cognitive investment required to manually read through the text of the paper is reduced.

Use Case 3

In the third use case, the researcher seeks information about *metadata* of one or more papers. In this use case, the researcher provides the KGQA system with *content* constraints about the papers and asks a question about the metadata of the paper. The KGQA system is then expected to extract metadata information related to the specific papers mentioned in the question.

As opposed to the first two use cases, the third use case is classified as *Adapted Condition Type: Content*. This is because the input constraint is based on content information. However, the expected answer is classified as *Adapted Answer Type: Metadata*, as the answer is expected to only contain metadata information.

A practical example of this use case would be a researcher asking for publications that contain a specific evaluation method or treat the same research problem. Such a question would require inferring over the contents of publications to find out whether they have applied the provided evaluation method or whether they treat the same research problem. In this case, the expected output would be the titles of the publications, which is metadata information, whereas the input constraint is based on the type of expected answer.

This use case demonstrates the potential for improving the current document-centric literature search workflow. This is because a major difficulty faced by researchers is the identification of relevant publications that contain specific information. In the current workflow, researchers have to rely on metadata during their search. However, the desired information is often not explicitly stated in the title or abstract of the publication. Therefore, it is difficult to find the relevant publications using traditional search engines. Consequently, a KGQA system that is able to find publications that contain specific content information would be a great improvement for the current literature research workflow.

Use Case 4

In the fourth use case, the researcher seeks information about the *content* of one or more papers. In this use case, the researcher provides the KGQA system with content information about the paper, and asks a question about the *content* of the paper. The KGQA system is then expected to extract content information related to the specific papers mentioned in the question.

The fourth use case has both the *Adapted Condition Type* and *Adapted Answer Type* classified as *Content*. This is because both the input constraints and expected output types are based on content information.

A practical example of this use case would be a researcher asking for common reference architectures that are proposed when tackling a specific research problem. In this case, the input constraint would be the research problem, while the expected output would be the reference architectures that are proposed in the literature. Both the input and output are content information, as they require reasoning over the content of the publications to find out which reference architectures are proposed for a specific research problem.

This use case is of particular interest when searching for definitions or explanations of specific concepts. In such cases, the location of the information in question is often uncertain, prompting the need to identify both the description and its source for the purpose of referencing it in another academic thesis.

Use Case 5

In the fifth use case, a researcher seeks information about the *content* of one or more papers. In this use case, the researcher provides the retriever with both *metadata and content* information about the papers and asks a question about the content of the paper. The retriever is then expected to extract content information related to the specific papers mentioned in the question.

The use cases until now only had a single type of input constraint. However, in the fifth use case, the input constraint is a combination of both *metadata and content* information. This makes the task more difficult for the KGQA system. It is now required to search for two different constraint types that might even be stored in different locations in the underlying KG. The expected output is *Adapted Answer Type: Content* as the answer is expected to contain content information.

Practical examples include combinations of the second and fourth use cases. For example, a researcher could ask for a summary of conclusions that publications have drawn in a specific time frame for a specific research problem. In this case, the input constraint would be the research problem and the time frame, which are content and metadata information, respectively. The expected output would be the summary of conclusions that are drawn in the publications, which is content information.

This use case is particularly interesting in scenarios where a researcher wants to save time by not reading the full text of a publication. Instead of reading the text, the researcher can pose questions to the document in order to retrieve the necessary information.

Use Case 6

In the sixth use case, the researcher seeks information about *metadata* of one or more papers. In this specific use case, the researcher provides the retriever with both *metadata and content* information about the papers, such as the name of an evaluation method and the year of publication, and asks a question about the metadata of the papers. The retriever is then expected to extract metadata information related to the specific papers mentioned in the question.

The sixth use case has the same *Adapted Condition Types* as the fifth use case, but the second dimension is *Adapted Answer Type: Metadata* as the answer is expected to contain metadata information.

The practical examples of this use case are a mixed version of the first and third use cases. For example, a researcher could request publications that have applied a specific evaluation method in a specific research field. In this case, the input constraint would be the evaluation method and the research field, which are content and metadata information, respectively. The expected output would be the titles of publications that have applied the evaluation method in the research field, which is metadata information.

Note that, as evident from the aforementioned use cases, the Answer Type dimension is not being mixed. According to the KGQA retrieval taxonomy, such a question would be designated as a Multiple Intentions question. Splitting this question into two distinct questions is a viable approach to enhance clarity and manage complexity. However, in the interest of maintaining manageable complexity, the multiple intentions are not being considered in the use cases.

8.3.2. Overview of Content Granularity

When creating datasets, it is useful to classify the level of granularity the data contains. This helps to provide a better understanding of the content of the dataset. For this purpose, we introduce four levels of granularity, which differ in their degree of abstraction:

1. **Document-based:** The current standard for distributing literature online. Publications are stored as PDF files in a database together with their metadata. Users locate publications using the metadata and then extract relevant information from the documents themselves by reading the provided texts.
2. **Chunk-based:** Originating from RAG [8], this approach divides documents into smaller chunks and embeds them into a vector space. Relevant chunks are retrieved from the database using nearest-neighbor search queries.
3. **Statement-based:** Here, concise summaries or statements are extracted from publications and stored directly in a database.
4. **Label-based:** The highest level of abstraction, where publications are classified using labels, which may include hierarchical structures. These labels give readers a quick overview of the content of a publication.

Using this classification, we assign the datasets that we used for our experiments to the *Label-based* abstraction level. This is because they consist of extracted terms structured according to a predefined schema as defined in Table 7.1.

8.3.3. Dataset Creation Process

Contribution **C2** of this thesis are KGQA datasets, which we used to carry out the experiments. These KGQA datasets were created using the data from the publication dataset introduced in Section 7.3.2. For their creation, the data was first loaded into the ORKG, and then KGQA pairs were generated using the semi-automated KGQA generation strategy described in Section 7.1.8. In the following, we present the creation process and the key characteristics of these datasets.

8.3.3.1. Dataset Structure

The datasets consist of KGQA pairs. Each pair includes the question itself together with the corresponding *ground-truth* data. This data is used during the evaluation to determine whether the answer and the data retrieved by the KGQA approach are correct. In this context, the ground truth represents one possible valid answer to the question, as well as a collection of triples needed to answer it. In addition to the question and the ground truth, each KGQA pair also includes further metadata. This encompasses a *topic entity* that can serve as an entry point into the graph for the retriever, the DOIs of the papers referenced in

the question, and the template used to generate the question. The pairs are also classified according to use case, semi-typed nature, and the categories defined in the taxonomy.

8.3.3.2. Question Diversity

To ensure that the questions in the KGQA datasets exhibit a high degree of variability that allows meaningful conclusions about the capabilities of a KGQA system, we structured the datasets considering multiple dimensions. First, we use the *use cases* described in Section 8.3.1 to map each question to a realistic application scenario. During the creation phase, we assigned questions to each of the six use cases to ensure a balanced distribution between them. To further reflect the retrieval capabilities required for each question, we incorporated the *Retrieval Operation* category from the KGQA retrieval taxonomy introduced in Chapter 6. Each question is assigned to a specific retrieval operation, ensuring that operations are covered evenly.

Additionally, the dataset distinguishes between *untyped* and *semi-typed* questions. This distinction is meant to assess how well a retriever can handle synonyms or missing type information:

- **Semi-Typed Questions** Each question in the dataset targets specific triples in the KG, with each triple consisting of a subject, predicate, and object. Depending on the triple, either the subject or the predicate may carry type information about the object. For example, the requested triple $(ResearchObjectEntity, Name, ReferenceArchitecture)$, requires the question to specify that the reference architecture is a research object.
- **Untyped Questions** In contrast, *untyped* questions do not include the type information. Using the same example, the question would simply ask about the reference architecture without stating that it is a research object. This increases the difficulty of retrieval because the retriever must infer the object type based solely on the object name.

8.3.3.3. Restrictions

To keep the datasets manageable in terms of complexity, certain constraints were applied during the creation. First, the number of golden triples needed to answer a question was limited to a maximum of 10. This limitation ensures that questions do not require information spread across the entire dataset. Without this restriction, answering broad questions would require extensive aggregation, resulting in runtimes that are too long for the scope of a master thesis.

Furthermore, it is essential that the retriever parameters are set in such a way that the retrievers can fully answer the given questions. By limiting the number of golden triples, we can ensure that this is the case. Consequently, the chosen threshold of 10 represents a compromise between the efficiency of the retrievers and the expressiveness of the questions. A higher threshold would require increasing retriever parameters, such as maximum depth

	Use Cases					
Retrieval Operation	**1**	**2**	**3**	**4**	**5**	**6**
aggregation	4	4	4	4	4	4
basic	4	4	4	1	4	1
comparative	4	4	4	4	4	4
counting	4	4	4	4	4	4
negation	0	0	4	4	4	4
ranking	4	4	4	4	4	4
relationship	4	4	4	4	4	4
superlative	0	0	4	4	4	4

Table 8.1.: Distribution of question templates across the use case and retrieval operation dimensions.

or width, leading to longer runtimes. A lower threshold, on the other hand, would reduce the informativeness of the questions or even make some of them unanswerable. Therefore, selecting a limit of ten triples strikes a balance between maintaining acceptable runtimes and preserving the relevance and feasibility of the questions.

8.3.3.4. Template Questions

To utilize the semi-automatic KGQA generation described in Section 7.1.8, template questions are required. For the datasets, a total of 170 different template questions were created manually, the full list of which is available in our replication package (see [124]). As mentioned previously, these questions were diversified along the dimensions of the use case and retrieval operation. The precise distribution of these template questions is shown in Table 8.1. For most combinations of use case and retrieval operation, four template questions were generated, although there are some exceptions. Specifically, for the use cases four and six, only two suitable examples were found for the *Basic* operation. Additionally, for use cases one and two, no suitable template questions were identified for the *Negation* and *Superlative* classes that were meaningful and below the threshold of ten golden triples.

Regarding the semi-typed nature of the questions, the dataset contains 86 semi-typed and 84 untyped questions. Generally, an equal number of semi-typed and untyped questions were created for each combination of use case and retrieval operation, with the exception of the individual questions for use cases four and six as mentioned above.

8.3.3.5. Classification According to KGQA Retrieval Taxonomy

According to the classification following the KGQA retrieval taxonomy described in Chapter 6, the distribution is as follows: Within the *Condition Type* category, 133 questions were classified as *Named Entity* and 37 as *Named Entity, Temporal*. This indicates that two types of

constraints must be considered by the retriever: either solely the consideration of a named entity or additionally a temporal constraint.

For the *Answer Format* category, the dataset comprises 61 *Enumerative*, 58 *Simple*, and 51 *Explanatory* questions. The classification according to *Graph Representation* shows that 152 questions belong to the *Multi-Fact* type, meaning they require multiple triples for their answer, while 18 questions are categorized as *Single-Fact*, requiring only one triple.

In the *Answer Type* category, 86 questions expect an answer of type *Named Entity*, 20 questions require a *Quantitative* answer, two questions anticipate a *Boolean* answer, and two additional questions have an *Other Type* of response. Furthermore, the dataset includes questions with more complex answer types: 24 questions expect either *Description* combined with *Quantitative* (*Description, Quantitative*) or *Description* with *Named Entity* (*Description, Named Entity*). Additionally, 13 questions require answers of the type *Description, Quantitative, Temporal*, and 8 questions expect answers of the type *Description, Named Entity, Temporal*, which means that they also require a year.

With respect to the *Intention Count* category, the dataset exclusively contains *Single Intention* questions, as multiple intentions were not considered. For the *Question Goal* category, due to insufficient variability in the publication data, only the class *Information Lookup* applies. Regarding *Answer Credibility*, all the questions meet the *Objective* criterion.

8.3.3.6. Dataset Creation

To generate the datasets, we used the semi-automatic cluster and subgraph-based generation methods introduced in Section 7.1.8. Based on the template question and the provided generation methods, we generated a total of 170 initial questions for the graph variant **GV2**. Then, each question and the corresponding ground truth were reviewed manually to ensure that both the question and the generated golden answers are consistent and grammatically correct. To further ensure quality, an additional LLM-based feedback script was employed, which checked each question and answer to provide feedback for changes if necessary.

After the quality of each question-answer pair was verified, we manually classified each question with the remaining categories of the KGQA retrieval taxonomy. Then, we employed the *conversion* scripts introduced in Section 7.3.5 to generate the KGQA datasets for the graph variants **GV1**, **GV3**, and **GV4**.

8.4. Evaluation Framework and Metrics

The KGQA approaches that we are testing in our thesis conform to the principles of RAG, because they retrieve information that is then used to improve the generation of an answer. Consequently, we employ an evaluation framework for RAG-based systems. A popular framework is Retrieval, Generation, and Additional Requirement (RGAR) [69], which will serve as the basis for structuring our evaluation. The framework is divided into three

modules. The first module, called *Evaluation Target*, defines the direction of the evaluation. The second module is *Evaluation Dataset*, which outlines how the datasets were created and selected. The last module is *Evaluation Metric*, which presents the specific metrics used to evaluate the targets based on the datasets.

In the following, we first introduce the evaluation targets differentiating between retrieval and generation objectives. Then, we briefly explain the datasets that were applied in the experiments. Finally, we introduce the GQM plan that shows how each target is evaluated using specific questions and metrics.

8.4.1. Evaluation Targets

This section introduces the evaluation targets, which are the constructs intended to be evaluated during the evaluation. Each evaluation target is defined using Evaluable Outputs (EOs) and Ground Truths (GTs).

8.4.1.1. Outputs and Ground Truth

EOs are the actual outputs of the KGQA approach, while GTs are the expected outputs as defined in the KGQA dataset. We define the following EOs and GTs for our evaluation:

EO1 Retrieved Triples

EO2 Generated Answer

GT1 Golden Triples

GT2 Golden Answer

Each KGQA approach returns two types of outputs in our setting. The first output is **EO1**, which represents the triples that have been retrieved from the ORKG and rated by the approach as relevant to answering the provided question. Second, **EO2** is the answer that has been generated by an LLM based on the retrieved triples and the question.

To evaluate the outputs of the retriever, the KGQA dataset provides two types of ground truth data. First, **GT1** are the golden triples, which are exactly those triples that are required to fully answer the question that was asked. Consequently, the retriever has to find all golden triples to be entirely successful at the retrieval task. Second, **GT2** is a golden answer formulated in natural language, which provides a response to the question and is based on the golden triples.

8.4.1.2. Evaluation Targets

The combination of EOs and GTs generates the evaluation targets, which are subdivided into targets for retrieval (*ReT*) and for generation (*GeT*):

ReT1 EO1 ↔ GT1

GeT1 EO2 ↔ GT2

GeT2 EO2 ↔ Question

GeT3 EO2 ↔ GT1

The targets introduced above guide the evaluation process. The retrieval target (**ReT1**) assesses the effectiveness of the context retrieval phase by comparing the retrieved triples (**EO1**) with the golden triples (**GT1**). This evaluation focuses on accuracy, relevance, and robustness. Here, accuracy is about the extent to which the retrieval approach is capable of fetching the desired triples (**GT1**) from the graph. Relevance refers to the degree to which the retrieved triples (**EO1**) correspond to the golden triples (**GT1**), where more irrelevant triples result in lower overall relevance. Finally, robustness refers to the ability of the retrieval system to maintain consistent performance in terms of accuracy and relevance despite variations encountered in the complexity of input questions (e.g., phrasing, the presence or absence of explicit type information, diverse use case requirements, and structural or lexical variability) across different knowledge graph schemas. The general intent behind **ReT1** is therefore to quantify how well the retriever identifies the correct information and how resilient its performance is under various operational conditions.

With regard to the generation targets, **GeT1** evaluates the quality of the generated answer by assessing its semantic and factual alignment with the reference ground truth answer (**GT2**). This target aims to measure how closely the generated answer of the system (**EO2**) matches an ideal and correct answer derived from the ground truth. The focus is on both the meaning and the accuracy of the information presented. High performance on this target indicates that the system can produce answers that are similar to a known correct answer.

GeT2 evaluates how well the generated answer (**EO2**) aligns with the intent and content requirements of the input question. This involves assessing whether the answer directly addresses the topic posed by the question and fulfills any specific instructions or constraints given, such as performing a ranking or counting operation. Consequently, this target measures the usefulness and appropriateness of the generated answer in the context of the question.

Finally, **GeT3** evaluates the faithfulness and adherence to the source material of the generated answer (**EO2**) relative to the retrieved triples (**EO1**). It evaluates whether the statements and claims made in the generated answer accurately and exclusively reflect the information present within the retrieved triples.

8.4.2. Evaluation Datasets

To evaluate the defined targets, datasets are needed to conduct the experiments. In Section 8.3 we document the creation of KGQA datasets. These datasets have been used to conduct the experiments and evaluate the targets.

8.4.3. Evaluation Plan

This section details the evaluation plan designed to assess the performance of the proposed HubLink system against baseline approaches. Following the introduction of the evaluation targets in Section 8.4.1, we adopt the GQM method [121, 122]. Consequently, we present a detailed GQM plan that systematically links each evaluation target (serving as a goal) with specific questions, which are in turn addressed using a defined set of quantitative metrics. To aggregate the metrics across all questions, we are using the *macro-averaging* strategy (see Section 2.6) because it attributes equal weight to each individual question without favoring those questions that request more triples.

In the following, we present each of the goals with their corresponding set of questions and metrics. We begin with the retrieval, followed by the generation.

8.4.3.1. Evaluation of the Retrieval Performance

The initial evaluation target, **ReT1**, centers on assessing the relevance and robustness of the contexts retrieved by HubLink. Several questions arise from this goal, evaluated using a suite of seven metrics. Standard information retrieval metrics, namely *Precision*, *Recall*, and *F1* are calculated based on the formulas presented by Yu et al. [69]. The determination of TP, FP, and FN involves comparing the triples retrieved by the system with the ground-truth triples provided within the reference KGQA dataset. The *Accuracy* metric is omitted from this evaluation, as its calculation requires knowledge of TN, which requires a complete listing of all irrelevant contexts, which is not available in our KGQA datasets.

To evaluate the ability of the retriever to rank relevant contexts highly, rank-aware metrics are incorporated. Specifically, *MAP@k* and *MRR@k* are adapted from the implementations of Tang and Yang [147]. The *Hits@k* metric is implemented following the definition provided by the AmpliGraph library[3]. In addition, *EM@k* is included, calculated according to the definition of Ibrahim et al. [72].

With regard to the required parameter k that determines the number of top contexts considered, we have set this parameter to 10. This choice reflects the maximum number of ground truth triples associated with any single question in our KGQA datasets, as detailed in Section 8.3.3.

[3] `https://docs.ampligraph.org/en/1.2.0/generated/ampligraph.evaluation.hits_at_n_score.html` [last accessed on 17.01.2025]

Furthermore, we test the environmental impact of the retrieval using the absolute, relative, and delta carbon emissions using the metrics specified by [78]. The tracking of the emissions is facilitated by the Codecarbon library[4], which tracks the *CPU Energy Consumption*, *GPU Energy Consumption*, and *RAM Energy Consumption*, measured in watts and then converts those measurements to estimate the total *Carbon Emissions* as CO_2 equivalent. Moreover, to evaluate runtime and token costs, we measure *System Latency* as the total time taken to retrieve the context and generate an answer while also tracking the number of *LLM Tokens* consumed during the retrieval and generation phases.

8.4.3.2. Evaluation of the Generation Performance

Transitioning to answer generation, the target **GeT1** focuses on comparing the factual and semantic similarity between the generated answer of the KGQA approach and the golden answer in the KGQA dataset. Lexical similarity is measured using *BLEU*[5] and *ROUGE*[6], employing their respective standard Python implementations. Semantic similarity is assessed using *BertScore*[7] utilizing the official implementation that provides Precision, Recall, and F1 variants based on contextual embeddings. Further evaluation leverages LLM-as-a-judge metrics provided by the RAGAS framework[8]. This includes *Factual Correctness*, where an LLM decomposes both the generated and reference answers into claims to calculate Precision, Recall, and F1 based on the comparisons of those claims [74]. We further use the *Semantic Similarity* metric, which calculates the cosine similarity and the *String Similarity* metric, which calculates the Levenshtein distance between the generated answer and the golden answer. Both metrics are also provided by the RAGAS framework.

The subsequent generation target, **GeT2**, addresses the alignment of the generated answers with the intent of the question, encompassing both semantic content and instructional constraints. This target involves two specific questions. The first, referring to semantic alignment, is evaluated using the *Answer Relevancy* metric from RAGAS. This metric uses an LLM to generate plausible questions based solely on the generated answer and then measures the semantic similarity (cosine similarity) between these generated questions and the original question [74]. The rationale is that a relevant answer should allow an accurate reconstruction of the original query. The second question, which focuses on adherence to instructional intent (e.g., performing a ranking task as requested), is assessed using a custom implementation that we developed on our own. In this process, an LLM evaluates the generated answer against the specific instructions embedded within the question to determine the degree of compliance.

The final generation target **GeT3** concerns the faithfulness of the generated answer to the information contained within the retrieved contexts. This is evaluated using the *Faithfulness* metric from the RAGAS framework. This metric utilizes an LLM to break down the generated

[4] `https://github.com/mlco2/codecarbon` [Accessed: 2025-01-17]
[5] `https://pypi.org/project/sacrebleu/` [last accessed on 17.01.2025]
[6] `https://pypi.org/project/rouge-score/` [last accessed on 17.01.2025]
[7] `https://github.com/Tiiiger/bert_score` [last accessed on 17.01.2025]
[8] `https://github.com/explodinggradients/ragas` [last accessed on 17.01.2025]

answer into individual statements and verifies each statement against the provided retrieved context [74]. This means that the metric evaluates whether the answer was actually derived from the context or from the internal knowledge of the LLM. We use this to evaluate whether the triples that were retrieved are actually reflected in the generated answer.

Goal-Question-Metric Plan:

Ret1 Assess the relevance and robustness of retrieved contexts in scholarly literature search.

Q1 To what extent does the HubLink retrieval algorithm improve context relevance and accuracy compared to baseline KGQA methods in scholarly literature search?

M1.1 Precision, **M1.2** Recall, **M1.3** F1, **M1.4** Hits@k, **M1.5** EM@k, **M1.6** MRR@k, **M1.7** MAP@k

Q2 How does retrieval performance vary with the complexity of operations required by different scholarly questions?

(see Q1.1)

Q3 How does retrieval performance vary across distinct scholarly literature-search use cases?

(see Q1.1)

Q4 What impact does the presence or absence of explicit type information in questions have on the retrieval performance?

(see Q1.1)

Q5 How robust is the proposed approach to structural and lexical variability across alternative knowledge graph schema representations?

(see Q1.1)

Q6 How efficient is the proposed approach considering runtime and language model tokens required when compared to baseline KGQA methods?

a) **M1.8** Runtime per question, **M1.9** LLM tokens per question

Q7 How does the proposed approach compare with regard to the environmental impact when compared to baseline KGQA methods?

a) **M1.10** Absolute carbon emissions, **M1.11** Relative carbon emissions, **M1.12** Delta carbon emissions

GeT1 Evaluate how well the generated answer aligns semantically and factually with reference answers.

Q8 How semantically and factually consistent are the generated answers of the proposed approach when compared to answers generated by baseline KGQA approaches?

M2.1 BLEU, **M2.2** ROUGE, **M2.3** Semantic Similarity, **M2.4** String Similarity, **M2.5** Bert Precision, **M2.6** Bert Recall, **M2.7** Bert F1, **M2.8** Factual Correctness Precision, **M2.9** Factual Correctness Recall, **M2.10** Factual Correctness F1

GeT2 Evaluate how well the generated answer aligns with the intent and content of the question.

Q9 To what extent do the answers generated by HubLink reflect the semantic intent of scholarly questions when compared to baseline KGQA approaches?

M3.1 Answer Relevancy

Q10 To what extent do the generated answers follow the instructional expectations of scholarly questions when compared to baseline KGQA approaches?

M3.2 Instruction Following

GeT3 Evaluate how well the generated answer aligns with the retrieved context.

Q11 To what extent are generated answers of HubLink faithful to the retrieved context and free from unsupported claims when compared to baseline KGQA approaches?

M4.1 Faithfulness

8.5. Dependent and Independent Variables

This section describes the independent and dependent variables of our experiments.

Independent Variables: These variables encompass the specific experimental configurations that are systematically varied between runs while performing the evaluation. The key independent variables include the choice of the core retrieval algorithm, modifications to its parameters, the specific dataset used for evaluation, the underlying knowledge base, and the selected LLM. These variables represent the factors that are manipulated to observe their effect on overall performance.

Dependent Variables: The dependent variables represent those constructs that we are interested in improving. Consequently, these constitute the quantitative performance measures used to assess the outcomes of different experimental configurations and directly correspond to the metrics detailed in the evaluation plan (see Section 8.4.3)

The experimental methodology involves the execution of a series of tests in which specific configurations of independent variables (pipeline components and parameters) are applied. For each configuration, the dependent variables (retrieval, generation, and operational metrics) are measured. The primary objective of this process is to analyze the collected data to determine the influence of variations in the independent variables on the observed system performance in the different categories of dependent variables. Because each KGQA

approach has a different set of parameters (independent variables), we employ a *Parameter Selection Process* to choose the appropriate values for our subsequent experiments. The selection process is documented in the following Chapter 9.

9. Parameter Selection Process

The HubLink and the baseline approaches each expose a variety of parameters that, in an ideal scenario, would be optimized via a full hyperparameter-tuning procedure using separate training and test datasets. However, we deviate from this optimal approach due to practical constraints, as we found that experiments with LLMs are both costly and time-consuming. Since the durations of multi-week experiments fall outside the scope of this master thesis, we adopt an alternative workflow we refer to as the *Parameter Selection Process*. The goal of this process is to determine the parameter configurations that optimize the retrieval performance of the KGQA retrievers in our experimental setting while respecting the time and resource constraints of this project. Consequently, the results of this process are configurations for each retriever, which we then used in our experiments.

The following sections are organized as follows. First, in Section 9.1, we explain the planning process that was done prior to conducting the selection process. Then, Sections 9.2, 9.3, 9.4, and 9.5, describe the selection process for HubLink and the baseline retrievers: DiFaR, FiDeLiS, and Mindmap. Finally, in Section 9.6, we discuss threats to the validity of the parameter selection process.

9.1. Planning

This section documents the planning of the parameter selection process. We first describe the methodology that was applied to realize the process. Next, we explain why we chose the Recall and Hits@10 metrics for the selection. Following this, we describe the dataset that was applied and the LLMs. Then, we describe the preliminary pipeline steps that are implemented before we describe why we decided to omit the StructGPT and ToG KGQA approaches.

9.1.1. Methodology

To carry out this process, we apply the One-Factor-at-a-Time (OFAT) method to identify the best parameters according to the Recall and Hits@10 metrics. In this method, a base configuration is selected and then each factor is successively varied over its range while all other factors are kept constant [148]. Consequently, we defined a base configuration and a range of parameter values for each KGQA approach. Then, following the OFAT method, multiple run configurations have been created based on the base configuration and ranges to subsequently test them using a reduced version of the KGQA dataset.

9.1.2. Metrics for Selection

To conduct the selection, metrics need to be chosen that allow us to determine whether a change in the value of a parameter is justified. For the selection process, we focused on retrieval performance rather than generation. This is because the generation of the answers depends on the retrieved contexts, which means that if the retriever cannot find the contexts that are relevant, it will not be able to generate an answer based on it. With regard to the choice of retrieval metrics, our primary metric is Recall, although we also use Hits@10 where needed. There are several reasons for these metrics, which we explain in the following:

First, when constructing the KGQA datasets (see Section 8.3), we ensured that only those triples were designated as the ground truth for which the content required to answer the question is actually present. We deliberately omitted any intermediate triples that must be traversed to reach the target since they are not strictly necessary to answer the actual question. To illustrate, consider a question that is asking for the authors of a paper. The authors are stored in separate nodes that all connect to a root node named *Authors List*. That root node serves only as a means to reach the author nodes and does not provide the information necessary to answer the question. In our preliminary tests, we observed that retrievers tend to include such intermediate nodes in their retrieved context. Furthermore, because answer generation occurs after context retrieval, these nodes could be helpful during answer generation by providing additional context to the LLM. Consequently, we chose not to penalize this behavior, which is consistent with using the Recall metric.

With regard to the Hits@10 metric, this metric allows us to understand whether the retriever ranks those contexts that are more relevant higher than those that are less relevant. For example, if the retriever includes intermediate triples in their output, the Hits@10 metric is still maximal if those triples that are actually relevant are higher on the list of outputs.

9.1.3. Choosing a Graph Variant

In Section 7.3.3 we introduced four different graph variants for our experiments to test the robustness of KGQA approaches. However, due to cost reasons, we were unable to run the parameter selection process (and experiments) on all four graph variants. We therefore decided on **GV1** as we believe that it is the graph variant with the most realistic modeling for real-world scenarios. The reason for this is that long paths allow the relationships between information to be captured, which preserves crucial context. Furthermore, the content is distributed by concern, which allows for extensibility in the future.

9.1.4. Using a Reduced KGQA Dataset

In addition to only running the selection process on one graph variant as mentioned above, we used a reduced version of the label-based KGQA dataset (for **GV1**) during the selection process. As described in Section 8.3, the KGQA datasets were created with

respect to use cases and retrieval operations. Each question is also classified as either semi-typed or untyped. For almost any pairing of a use case with a retrieval operation, there are four corresponding pairs of questions and answers. When constructing the reduced dataset, we therefore selected one semi-typed question per combination to ensure that each question is representative of the larger dataset. We chose semi-typed over untyped questions, as we expected them to perform better, which is important when selecting parameters. Consequently, the reduced KGQA dataset for graph variant **GV1** contains a total of 44 questions.

9.1.5. Large Language and Embedding Models

As all retrievers are based on LLMs, the model selection is crucial for the performance of the retriever. Since HubLink, DiFaR, Mindmap, and FiDeLiS also work with embeddings, the selection of the embedding model is equally important.

For our experiments and the selection process, we implemented the following endpoints: *OpenAI* as a proprietary provider as well as *Ollama* and *Hugging Face* for open-source models, both of which are run locally on the server. Furthermore, when choosing which models to use, we considered the following points:

1. The OpenAI endpoint is proprietary and can introduce high costs if not managed carefully. As such, we considered the associated costs of the models and how many models from OpenAI we are using.
2. Through testing, we found the Hugging Face models to be less optimized than the Ollama ones. This means that the amount of hardware memory resources required to run models on the Hugging Face endpoint is higher than on the Ollama endpoint, which may lead to *out-of-memory* errors.
3. We are restricted to the hardware resources available on our server. We have $32GB$ of GPU memory available, which is enough to fit optimized LLMs of the size of $32B$ parameters on the GPU. However, running embedding models in parallel is then not feasible. Moreover, even if a large model fits on the GPU, its response time is likely too slow to be used in our experiments. Consequently, we chose to use smaller models.

To help in the selection process, we reviewed popular leaderboards to assess the performance of the models available. We examined two leaderboards, both reflecting the status as of February 16, 2025. For LLMs, we examined the *Chatbot Arena Leaderboard*[1], proposed by Chiang et al. [149]. For embedding models, we observed the *Massive Multilingual Text Embedding Benchmark (MMTEB)*[2], introduced by Enevoldsen et al. [150]. A snapshot of both leaderboards at the time of review is available in our replication package [124].

[1] `https://huggingface.co/spaces/lmarena-ai/chatbot-arena-leaderboard` [last accessed on 16.02.2025]
[2] `https://huggingface.co/spaces/mteb/leaderboard` [last accessed on 16.02.2025]

9.1.5.1. Selection of LLMs

We selected the following LLMs for our experiments: *GPT-4o*, because the model is ranked at the highest position in the Chatbot Arena leaderboard via the OpenAI endpoint. *GPT-4o-mini*, ranked 24th yet delivering a strong performance at a fraction of the cost and also *O3-mini*, a newly released model that inherently implements chain-of-thought reasoning [64]. To include open-source options, we chose *Qwen2.5*, which is the Ollama endpoint model that performs the best on the leaderboard. However, due to our hardware constraints, we had to reduce the model to its 14*B* parameter variant. Furthermore, we selected *Llama3.1*, which represents the second-best Ollama model in the leaderboard. However, we had to scale it down to the 8*B* parameter model because of hardware constraints. We also evaluated *DeepSeek-R1* [151], a new open-source reasoning model with promising benchmarks, but its performance-to-runtime ratio was substantially worse than that of our selected models, so we excluded it.

9.1.5.2. Selection of Embedding Models

For embedding models, we included *text-embedding-3-large*, the largest embedding model available via the OpenAI API. With regard to open-source models, we chose the *Mxbai-Embed-Large* model, which is a fast and popular open-source model ranked 41st on the MMTEB leaderboard. Because it has a quick response time with good performance, it is a good choice for the base configurations in our selection process. We also evaluated *Granite-Embedding*, a new Ollama endpoint model that is not yet on the leaderboard. Still, it is a promising model that is fast and looks to have a good performance. Finally, we tested *gte-Qwen2-7B-instruct*, the top-ranked MMTEB model, but it exhibited slow inference and unexpectedly poor performance. We are not entirely sure why the performance of the model was poor, but we suspect that it may be due to the fact that it was used over the Hugging Face endpoint, which uses unoptimized models. Ollama, on the other hand, provides expert optimization for their models, which makes them faster and could make them perform better. This is the reason we opted to use models from Ollama over those provided on Hugging Face.

9.1.6. Pre-, Post-Retrieval and Generation

Our RAG pipeline involves four steps: 1) Pre-Retrieval, 2) Retrieval, 3) Post-Retrieval, and 4) Generation. In the following, we are going to introduce each step and its relevance to the parameter selection process.

Pre-Retrieval: The pre-retrieval step is responsible for the preprocessing of the input question. We implemented a question augmentation technique that prompts an LLM to improve the given question by clarifying ambiguities, incorporating related keywords or phrases that will help the retrieval system retrieve more accurate and comprehensive information, and adding nouns or noun phrases to terms to clearly indicate their types

or roles. Regarding the parameter selection, we tested each retriever with and without augmentation.

Retrieval: The retrieval step is where both HubLink and the KGQA baseline retrievers are applied. Each KGQA approach has its own set of parameters relevant to the parameter selection process. For each parameter, we chose a range of values that were tested. The ranges are documented for each approach in the following sections.

Post-Retrieval: In the post-retrieval step, the retrieved context from the previous step is processed. We implemented a function that prompts an LLM to rerank the retrieved contexts based on the provided question. During the parameter selection process, we then tested each KGQA approach with and without reranking.

Generation: The generation step is responsible for generating the final answer based on the question and the contexts that have been retrieved. The generation is done by prompting an LLM with the question and the contexts and asking it to generate an answer. However, because almost all KGQA approaches provide an answer as part of their procedure, the generation step is skipped to retain the original answer of the approach. The only exception is DiFaR, for which generation prompting is used.

We provide the prompts that have been used for the question augmentation, reranking, and generation procedures in Appendix A.1.

9.1.7. Omitting StructGPT and ToG

The use of the StructGPT [92] and ToG [88] KGQA approaches proved to be unsuitable in our experimental setting. Both approaches were unable to retrieve any relevant information from the graph, which is why we omitted them from the selection process and the experiments. A more detailed analysis of why these approaches are unable to answer the questions in our experiment can be found in Section 10.3.

9.2. Parameter Selection for HubLink

In the sections that follow, we initially outline the base configuration and the range of parameters explored during the parameter selection phase for the HubLink retriever. Subsequently, we analyze the results of the test runs and detail the parameter values chosen for the final configuration utilized in our subsequent evaluations.

9.2.1. Base Configuration and Parameter Ranges

In Table 9.1, the parameter values for the base configuration and the ranges of parameters tested for HubLink are presented, which we will discuss in the following:

Parameter	Parameter Space
`Do Traversal Strategy`	False, True
`Extract Question Components`	False, True
`Top Paths to Keep`	10, 20, 30
`Number of Hubs`	10, 20, 30
`Filter Output Context`	False, True
`Diversity Ranking Penalty`	0, 0.01, 0.05, 0.1
`Path Weight Alpha`	0, 3, 5, 9
`Do Question Augmentation`	False, True
`Do Reranking`	False, True
`LLM`	gpt-4o-mini, gpt-4o, o3-mini, Qwen2.5-14B, Llama3.1-8B
`Embedding Model`	mxbai-embed-large, text-embedding-3-large, granite-embedding

Table 9.1.: The base configuration (underlined) and the parameter space for HubLink.

Do Traversal Strategy: The parameter determines whether the approach can use a provided topic entity that accompanies the question. When set to `True`, the *graph traversal* strategy is employed, using the topic entity as the starting point in the graph. However, for the base configuration of the selection process, we used the *direct retrieval* strategy because this strategy is faster, allowing us to test more configurations during the selection process.

Extract Question Components: This parameter specifies whether the component extraction process is used. If used, the question components are extracted using an LLM and utilized in addition to the question during the search for relevant hubs. This technique is integral to the HubLink retriever and has the purpose of enhancing the performance of queries that have multiple constraints. To evaluate our design decision to add this extraction, we tested the difference in performance during the selection process when the extraction is enabled or disabled. For the base configuration, it remains enabled.

Filter Output Context: When generating partial answers, the hub paths are used. Each path can include multiple triples that form the path. To determine which of the triples are actually needed for the answer, we added a filtering based on an LLM. During the selection process, we tested our design decision of adding this filtering step by disabling it in one execution. However, for the base configuration the filtering remains enabled, as it is an integral part of the HubLink approach.

Top Paths to Keep: This parameter defines the number of paths retained per hub. Increasing this value allows more context to be used during partial answer generation, which can improve performance, but may also introduce noise. We set this value at 10, as this is the maximum number of triples requested in the applied KGQA dataset. We also increased the value to 20 and 30 to test the effect of allowing more context.

Number of Hubs: The parameter specifies how many hubs are used to generate partial answers. A higher value increases the chances of finding relevant context, but also increases runtime and cost. We set 10 as the starting point and varied the values in increments of 10 up to 30 to assess the impact of additional context on retrieval performance. As mentioned previously, the starting value is based on the maximum number of triples requested from the KGQA dataset.

Diversity Ranking Penalty: This parameter influences the overall score of each hub and determines the emphasis on diversity among the hub paths during reranking. This technique is a core part of HubLink and was added during the design phase to decrease the likelihood of hubs being pruned because they have a high diversity of scores. The higher the value, the more diversity is tolerated. During development, we found that the value of 0.05 provides satisfactory results, which is why we set it as the default. We further tested the values of 0, 0.01 and 0.1 to see how they change the results.

Path Weight Alpha: During the design of HubLink, we added weighting when calculating the final scores of the hubs before pruning them, which in theory should reduce the likelihood of pruning hubs that have unevenly distributed hub path scores. During development, we found the value of 5 to provide satisfactory results, which is why we set it as the value in the base configuration. We also tested the values 0, 3, and 9.

Do Question Augmentation: This parameter determines whether the question is augmented before it is provided to the KGQA approach. It is not part of the HubLink retriever itself but rather a pre-retrieval step that is applied before sending the question to the retriever. We disabled it by default and assessed its impact on the results during the parameter selection process.

Do Reranking: The reranking process is also not part of the approach but rather a post-retrieval step that is applied on the list of contexts returned by the approach. In theory, it should improve the results of rank-based scores, as those contexts that are relevant are put at the top of this list. However, if the approach is already good at ranking by relevance, adding this step should not have much of an effect.

LLM: The LLM is used to generate partial answers, filter the context, convert paths to texts, and generate the final answer. The models selected for the parameter selection process were already introduced in Section 9.1.5. For our base configuration, we used two models, *gpt-4o-mini* and *Qwen2.5-14B*, to reduce the runtime of the parameter selection process by running multiple configurations in parallel. Therefore, the embedding models and the other LLMs have been tested with the *Qwen2.5-14B* model as the base, and the remaining parameters have been run with *gpt-4o-mini* as the base.

Embedding Model: The embedding model is used to transform the question and the hubs into vectors. The models that we used during the selection process are introduced in Section 9.1.5. Regarding the base configuration, we used the *mxbai-embed-large* model because it is cost-efficient and fast.

Parameter	Value	Recall	Hits@10
Large Language Model	gpt-4o-mini	0.512	0.372
	gpt-o3-mini	0.608 (+18.7%)	**0.499 (+34.1%)**
	gpt-4o	**0.615 (+20.1%)**	0.481 (+29.3%)
	Qwen2.5-14B	0.448 (-12.5%)	0.367 (-1.3%)
	Llama3.1-8B	0.374 (-27.0%)	0.259 (-30.4%)
Embedding Model	mxbai-embed-large	0.448	0.367
	text-embedding-3-large	**0.555 (+23.9%)**	**0.494 (+34.6%)**
	granite-embedding	0.490 (+9.4%)	0.405 (+10.4%)
Path Weight Alpha	0	0.391 (-23.6%)	0.248 (-33.3%)
	3	0.491 (-4.1%)	0.340 (-8.6%)
	5	**0.512**	**0.372**
	9	**0.505** (-1.4%)	0.343 (-7.8%)
Diversity Ranking Penalty	0.00	0.338 (-34.0%)	0.265 (-28.8%)
	0.01	0.412 (-19.5%)	0.323 (-13.2%)
	0.05	**0.512**	**0.372**
	0.10	0.474 (-7.4%)	0.328 (-11.8%)
Top Paths to Keep	10	**0.512**	**0.372**
	20	0.438 (-14.5%)	0.301 (-19.1%)
	30	0.247 (-51.8%)	0.155 (-58.3%)

Table 9.2.: The results of the parameter selection process for HubLink. The base configuration parameter is underlined, and the highest metric score per parameter is **bold**. This is the first out of two tables that display the results.

9.2.2. Parameter Selection

We ran a total of 22 configurations that we split into two base configurations to run them in parallel. The final configuration parameters are shown in Table 9.4 and the results of the parameter selection process are presented in Table 9.2 and Table 9.3, which we will discuss in the following:

Large Language Model: Comparing the five LLMs, we observe a strong correlation between the capability of the model and the retrieval performance. The *gpt-4o-mini* model achieved a Recall of 0.512 and a Hits@10 of 0.372. Larger and more advanced models like *gpt-o3-mini* and *gpt-4o* significantly outperformed the baseline, achieving Recall improvements of +18.7% and +20.1% respectively, and Hits@10 improvements of +34,1% and +29.3%. The results also indicate that the open-source models, *Qwen2.5-14B* and particularly *Llama3.1-8B*, have considerably lower performance. This suggests that the capabilities of the LLM have a high impact on the effectiveness of HubLink, and potentially even better results could be achieved with larger or more capable future models. Although *gpt-4o* achieved the highest Recall (+20.1%), *o3-mini* reached the highest Hits@10 score (+34.1%). Therefore, both are

Parameter	Value	Recall	Hits@10
Number of Hubs	10	0.512	0.372
	20	0.517 (+1.0%)	0.325 (-12.6%)
	30	**0.554 (+8.2%)**	**0.356 (-4.3%)**
Output Filtering	True	0.512	**0.372**
	False	**0.631 (+23.2%)**	0.191 (-48.7%)
Extract Question Components	True	**0.512**	0.372
	False	0.440 (-14.1%)	**0.375 (+0.8%)**
Do Question Augmentation	True	0.500 (-0.2%)	0.333 (-8.8%)
	False	**0.512**	**0.372**
Do Reranking	True	0.497 (-2.8%)	**0.334 (-10.4%)**
	False	**0.512**	0.372
Do Traversal Strategy	True	**0.559 (+9.2%)**	**0.422 (+13.4%)**
	False	0.523	0.395

Table 9.3.: The results of the parameter selection process for HubLink. The base-configuration parameter is underlined, and the highest metric score per parameter is **bold**. This is the second out of two tables that display the results.

strong choices for the final configuration. We selected ***o3-mini*** for our final configuration because it is the newer model according to its release date.

Embedding Model: For the embedding model comparison, the baseline *mxbai-embed-large* resulted in a Recall of 0.448 and Hits@10 of 0.367. Switching to *text-embedding-3-large* produced substantial improvements across both metrics, boosting Recall by +23.9% to 0.555 and Hits@10 by +34.6% to 0.494. The *granite-embedding* model also outperformed the baseline (+9.4% Recall, +10.4% Hits@10) but was clearly inferior to *text-embedding-3-large*. This indicates that the choice of embedding model has a major impact on the performance of HubLink. Based on these results, we selected ***text-embedding-3-large*** as the embedding model for the final configuration.

Path Weight Alpha: Setting alpha to 0 resulted in a drastic performance drop (-23.6% Recall, -33.3% Hits@10), indicating that path weighting is essential. The baseline value of **5** achieved the highest scores for both Recall (0.512) and Hits@10 (0.372). Increasing alpha further to 9 led to a slight decrease in performance compared to the baseline. This suggests that while weighting paths is beneficial, excessive weighting might negatively impact the results. Consequently, we retained the baseline value of **5** for the alpha parameter.

Diversity Ranking Penalty: The results for the diversity penalty mirror those of the `PathWeightAlpha` parameter. Disabling the diversity penalty (0.00) resulted in a drastic performance drop (-34.0% Recall, -28.8% Hits@10). The baseline value of **0.05** yielded the best results, achieving the highest Recall (0.512) and Hits@10 (0.372) among the values tested.

Increasing the penalty to 0.10 resulted in lower scores compared to 0.05. This suggests that while diversification is beneficial, an excessive penalty might negatively impact results. We therefore selected **0.05** for the diversity penalty.

Top Paths to Keep: The baseline for the number of paths that are kept per hub was set to 10. Increasing the number of paths to 20 or 30 led to a significant drop in performance. Keeping 20 paths reduced Recall by 14.5% and Hits@10 by 19.1%, while keeping 30 paths caused a drastic drop (-51.8% Recall, -58.3% Hits@10). The baseline value of **10** yielded the best performance, which is unexpected because, in theory, adding more paths provides more context for each hub when generating partial answers, increasing the chance of retrieving relevant information from the graph. We hypothesize that this discrepancy is due to the use of the *gpt-4o-mini* model during execution. It is likely that the model was overwhelmed by the large volume of context, causing relevant information to be lost because of additional noise in the data. As a result, the model may have struggled to extract key facts from the data. It remains to be tested whether the results persist with a different LLM that handles large contexts more effectively. However, for our final configuration, we rely on the current results rather than assumptions and set the number of paths to 10.

Number of Hubs: The effect of increasing the number of hubs contrasts with the `TopPathsToKeep` parameter. While keeping more paths per hub degraded performance, increasing the number of hubs from the baseline of 10 improved Recall (+1.0% for 20 hubs, +8.2% for 30 hubs). This supports the intuition that providing more context increases the likelihood of retrieving relevant information. We hypothesize that this works better than increasing paths per hub because the context is processed sequentially by the LLM rather than simultaneously. However, the Hits@10 metric showed a less positive trend, decreasing for 20 hubs (-12.6%) and remaining slightly below the baseline even for 30 hubs (-4.3%). This suggests a trade-off where more hubs lead to more relevant information overall, increasing the Recall, but this might make it slightly harder to rank the relevant contexts within the top 10. For our final configuration, we prioritized the notable Recall improvement and selected **30** hubs, accepting the minor Hits@10 decrease relative to the 10-hub baseline.

Output Filtering: The output filtering step has the aim of reducing the HubLink output to the triples that are actually relevant to the question. Without filtering, HubLink returns all triples from the paths used in the final answer, including many that are not useful. The baseline configuration had filtering enabled, yielding Recall 0.512 and Hits@10 0.372. Disabling filtering resulted in a substantial Recall increase to 0.631 (+23.2%) but caused a drastic drop in Hits@10 to 0.191 (-48.7%). This clearly demonstrates the function of the filter as removing irrelevant triples boosts precision (increasing Hits@10) at the cost of potentially removing some relevant triples (reducing the Recall). Because we were using *gpt-4o-mini* for the baseline, we hypothesize that its limitations might contribute to removing relevant triples alongside irrelevant ones. Given that our final configuration uses the more capable *o3-mini* model, we anticipate its filtering performance will be improved. Therefore, despite the Recall drop observed with the baseline model, we **enabled** output filtering in the final configuration, prioritizing cleaner, more precise results.

Parameter	Value
LLM	gpt-o3-mini
Embedding Model	text-embedding-3-large
Do Traversal Strategy	True
Extract Question Components	True
Top Paths to Keep	10
Number of Hubs	30
Filter Output Context	True
Diversity Ranking Penalty	0.05
Path Weight Alpha	5
Do Question Augmentation	False
Do Reranking	False

Table 9.4.: The final configuration used in subsequent experiments for our proposed HubLink approach.

Extract Question Components: The goal of extracting components from questions is to help the retriever better handle complex queries with multiple constraints. Enabling this component extraction resulted in a Recall of 0.512 and Hits@10 of 0.372. Disabling it caused a substantial drop in Recall to 0.440 (-14.1%), although it yielded a marginally higher Hits@10 score of 0.375 (+0.8%). However, the significant impact on Recall outweighs the negligible gain in Hits@10. Therefore, we conclude that component extraction is crucial for effectiveness and **enabled** the extraction for the final configuration.

Do Question Augmentation: Disabling the prior augmentation of the question yielded a Recall 0.512 and Hits@10 0.372. Enabling the augmentation had a negligible impact, with Recall dropping to 0.500 (-0.2%) but resulted in a noticeable decrease in the Hits@10 score to 0.333 (-8.8%). Although question augmentation might be more beneficial for less structured or untyped queries, our experiments on the typed questions in the reduced QA dataset did not show an advantage. Based on the observed decrease in ranking performance without any Recall benefit, we **disabled** the augmentation for the final configuration.

Do Reranking: The inclusion of a ranking step, where the LLM reorders the initially retrieved paths based on relevance to the question, should improve the Hits@10 score. The results show that disabling the ranking resulted in a Recall of 0.512 and Hits@10 of 0.372. Enabling the reranker unexpectedly led to slightly worse performance on both metrics. Recall decreased slightly to 0.497 (-2.8%), and Hits@10 decreased to 0.334 (-10.4%). These results contradict the expectation that reranking should primarily boost Hits@10. The observed degradation suggests either that the specific reranking implementation or model used was ineffective for this task, or potentially that the initial ranking provided by HubLink was already close to optimal, and the reranking step introduced noise. Given that enabling reranking demonstrated worse performance, we **disabled** it for the final configuration.

Parameter	Parameter Space
`Distance Metric`	cosine, L2, IP
`Number of Results`	30, 60, 90, 120, 150
`Do Question Augmentation`	False, True
`Do Reranking`	False, True
`LLM`	gpt-4o-mini
`Embedding Model`	mxbai-embed-large, text-embedding-3-large, granite-embedding

Table 9.5.: The base configuration (underlined) and parameter space for DiFaR.

Traversal Strategy: This parameter switches HubLink from the *direct retrieval strategy* to the *graph traversal strategy*. The former achieved a Recall of 0.523 and Hits@10 of 0.395, while the latter achieved a Recall of 0.559 (+9.2%) and Hits@10 of 0.422 (+13.4%). Given these results, we observe a small increase in performance using the traversal strategy. This would suggest that it is helpful to provide a topic entity that guides the retrieval procedure. However, we argue that the KG that we are using for the experiments is too small to gather significant results. It remains to be tested on a large graph whether the retrieval substantially improves with the traversal strategy. For our final configuration, we **enabled** the traversal strategy due to the improvement in performance on our data.

9.3. Parameter Selection for DiFaR

In the sections that follow, we initially describe the base configuration and the range of parameters evaluated during the parameter selection process for the DiFaR approach. Subsequently, we analyze the outcomes of the test runs and outline the parameter values chosen for the final configuration employed in our later experiments.

9.3.1. Base Configuration and Parameter Ranges

In Table 9.5, the parameter values for the base configuration and the ranges of tested parameters for DiFaR are presented. In the following, we briefly introduce each parameter.

Distance Metric: The distance metric determines how the similarity score between the embeddings in the vector store and the question is calculated. Three distance metrics are supported: cosine, inner-product, and l2 distance. We tried each of the metrics in the selection process. The cosine distance was used in the base configuration because of its popularity.

Number of Results: This parameter determines how many triples are obtained from the vector store for the generation of the answers. The more triples are fetched, the more context is available for the LLM to generate the answer. However, this also increases the cost of the retrieval process and can introduce more noise into the data. We tested the values 30, 60, 90, 120, and 150 to see how the number of triples affects the performance. For the base configuration we set the value to 30.

Do Question Augmentation: This parameter determines whether the question is augmented before it is sent to DiFaR. As such, it is not part of the approach itself but rather a pre-retrieval step. We have disabled it in the base configuration and assessed its impact on the results when enabling it during testing.

Do Reranking: The reranking process is also not part of the approach but a post-retrieval step that is applied to the list of contexts returned by the approach. We also disabled it in the base configuration but ran a configuration where it was enabled.

LLM: The LLM is used to generate the final answer based on the retrieved contexts. Because it is not part of the retrieval process itself, it has no impact on the Recall and Hit@10 metrics, which is why we set the LLM to the *gpt-4o-mini* model, which is cost effective and fast.

Embedding Model: The embedding model, on the other hand, influences the retrieval process as it is used to generate the embeddings of the question and the triples. The embedding models were already introduced in Section 9.1.5. For the base configuration, we used the model *mxbai-embed-large* because it is cost effective and fast.

9.3.2. Parameter Selection

In Table 9.6 the results of 11 different configurations of the DiFaR approach are shown. In the following, we discuss the results and determine which parameters to use for the final configuration. The final configuration for the DiFaR approach is shown in Table 9.7.

Number of Results: As expected, increasing the number of retrieved contexts generally improves the probability of capturing the information sought after, leading to a higher Recall. Our results confirm this trend, with Recall steadily increasing from 0.300 at 30 results to 0.366 (+22.1%) at 150 results. Interestingly, the Hits@10 metric remained constant at 0.272 for 30, 60, 90, and 120 results and even saw a minor improvement to 0.288 (+5.6%) when retrieving 150 results. We set the value to 150 for the final configuration.

Parameter	Config	Recall	Hits@10
Number of Results	30	0.300	0.272
	60	0.310 (+3.4%)	0.272 (+0.0%)
	90	0.312 (+4.2%)	0.272 (+0.0%)
	120	0.322 (+7.4%)	0.272 (+0.0%)
	150	**0.366 (+22.1%)**	**0.288 (+5.6%)**
Embedding Model	mxbai-embed-large	**0.300**	**0.272**
	granite-embedding	0.276 (-7.8%)	0.254 (-6.8%)
	text-embedding-3-large	0.238 (-20.5%)	0.194 (-28.8%)
Distance Metric	cosine	0.300	0.272
	IP	0.300 (+0.0%)	0.272 (+0.0%)
	L2	0.300 (+0.0%)	0.272 (+0.0%)
Do Reranking	False	0.300	0.272
	True	0.300 (+0.0%)	0.278 (+2.2%)
Do Question Augmentation	False	**0.300**	**0.272**
	True	0.248 (-17.4%)	0.196 (-28.1%)

Table 9.6.: The results for the parameter selection process for the DiFaR approach. Here, the base configuration parameter is underlined, and the highest metric score per parameter is **bold**.

Large Language Model: As mentioned above, the LLM has no impact on retrieval performance, as it is only used to generate the answer after the contexts have been retrieved. We have set the number of retrieved triples to 150, which benefits Recall and eventually Hits@10, but also increases the amount of context fed into the answer generation component. This needs a strong model to successfully extract the necessary information. We therefore selected the model*gpt-o3-mini* for the final configuration, which we expect to be large enough to handle the context window and to provide good performance in extracting contexts due to its reasoning capabilities.

Embedding Model: The choice of the embedding model is critical for DiFaR, as the approach relies on a ANN search. The baseline model *text-embedding-3-large* achieved the best performance with Recall 0.300 and Hits@10 0.272. Interestingly, *text-embedding-3-large*, which performed strongly in other parts of our study, scored significantly lower here. The Recall is 0.238 (-20.5%) and Hits@10 is 0.194 (-28.8%). The *granite-embedding* model also underperformed compared to the baseline with with Recall 0.276 (-7.8%) and Hits@10 0.254 (-6.8%). Based on these results, we selected the model ***mxbai-embed-large*** for the final DiFaR configuration.

Distance Metric: We tested three common distance metrics for vector similarity: *Cosine*, *Inner Product (IP)*, and *Euclidean (L2)*. The results do not indicate any difference in performance. All three metrics yielded identical Recall (0.300) and Hits@10 (0.272) scores. Given

Parameter	Value
`Number of Results`	150
`Distance Metric`	Cosine
`LLM`	gpt-o3-mini
`Embedding Model`	mxbai-embed-large
`Do Question Augmentation`	False
`Do Reranking`	False

Table 9.7.: The final configuration used for subsequent experiments for the DiFaR KGQA baseline approach.

that no difference was observed, we retained the default distance metric ***cosine*** for the final configuration.

Do Reranking: Reranking the contexts after retrieval should not affect the Recall metric, which is also reflected in the results (0.300). However, in theory, it should improve the Hits@10 score if the approach returns the context without ranking by relevance. DiFaR inherently ranks results based on vector similarity to the query embedding. Theoretically, an additional reranking step could refine this order to further improve Hits@10. Our results show that enabling reranking led to a negligible increase in Hits@10 to 0.278 (+2.2%) compared to the baseline score of 0.272. Although this represents a small improvement, considering the added computational cost and complexity of a reranking step, we deem this gain insufficient to warrant its inclusion. Therefore, we decided to **disable** reranking in the final configuration.

Do Question Augmentation: Similar to our findings with HubLink, enabling question augmentation before querying the DiFaR approach significantly worsened performance compared to the baseline without augmentation. Both Recall (0.248, -17.4%) and Hits@10 (0.196, -28.1%) dropped significantly. This suggests that modifications introduced by the augmentation process might add noise or irrelevant terms that mislead the nearest-neighbor search in the embedding space. Consequently, we **disabled** the augmentation for the final DiFaR configuration.

9.4. Parameter Selection for FiDeLiS

In the following sections, we first introduce the base configuration and the parameter ranges that were tested in the parameter selection process for the FiDeLiS approach. Then, we discuss the results of the test runs and explain which parameter values were selected for the final configuration used in our subsequent experiments.

Parameter	Parameter Space
`Top k`	10, 20, 30
`Top n`	10, 20, 30
`Alpha`	0.1, 0.3, 0.6
`Do Question Augmentation`	False, True
`Do Reranking`	False, True
`LLM`	gpt-4o-mini, gpt-4o, o3-mini, Qwen2.5-14B, Llama3.1-8B
`Embedding Model`	mxbai-embed-large, text-embedding-3-large, granite-embedding

Table 9.8.: The base configuration (underlined) and the parameter space for FiDeLiS.

9.4.1. Base Configuration and Parameter Ranges

In Table 9.8, the parameter values for the base configuration and the ranges of tested parameters for FiDeLiS are presented. In the following, we briefly introduce each parameter.

Top k: This parameter determines the width of the beam search at each given step. The paths that are found are pruned by relevance with an LLM to k paths. As such, when the parameter value is increased, more contexts are gathered, which should increase the likelihood of finding relevant information. We started with a value of 10, as this is the maximum number of triples requested in the KGQA dataset and increased the value from there to 20 and 30 to see how the number of paths affects performance.

Top n: This parameter determines the number of neighbor paths that are kept when expanding each current path, which determines the number of paths that are sent to the LLM for subsequent pruning. The more paths are kept, the higher the likelihood of finding relevant information. Similarly to the k parameter, we started with a value of 10 and increased the value from there to see how it affects the performance.

Alpha: This parameter determines the weight that the path score gets in comparison to the relation and neighbor scores. The higher the number, the more weight the path score gets. We started from the proposed default value from the authors, which is 0.3, and varied the score to 0.1 and 0.6.

Do Question Augmentation: Same as with the other approaches, we disabled the question augmentation in the base configuration and assessed its impact on the results during testing.

Do Reranking: Same as with the other approaches, we disabled the reranking by default and assessed the impact of enabling the reranking in one of the test runs.

LLM: We expect the LLM to have a major impact on the performance of the approach as it guides the process of finding the relevant paths. We already introduced the models that were used in the parameter selection in Section 9.1.5. For our base configuration we used the model *gpt-4o-mini* as it is cost effective and fast.

Embedding Model: The embedding model is used to generate the embeddings of the keywords from the question, as well as the predicates and entities from the graph. We also expect this to have a major impact on the performance as this is the primary way of pruning the paths in each depth of the beam search. We used *text-embedding-3-large* as the embedding model for our base configuration, as it allows to run multiple RAG pipelines in parallel because the model is accessed over an API and not run locally, circumventing hardware constraints. The other models that were tried are introduced in Section 9.1.5.

9.4.2. Parameter Selection

The results for 13 different configurations of the FiDeLiS approach are presented in Table 9.9. It is immediately apparent that the overall performance scores are considerably lower for FiDeLiS compared to the other approaches tested, indicating challenges with this specific question-answering task, which we discuss in Section 10.3. Because the scores are rather low, it is hard to attribute the observed results to the change of the parameter or the randomness involved in the retrieval process. While we proceed with parameter selection based on the available data, the choices have been made with low confidence due to these factors. The final configuration for the FiDeLiS approach is shown in Table 9.10.

Large Language Model: Looking at the results, the choice of LLM has significantly impacted the performance of FiDeLiS. The baseline model, *gpt-4o-mini*, achieved Recall and Hits@10 scores of 0.129. The *gpt-o3-mini* model performed slightly better, yielding scores of 0.136 (+5.4%). The *gpt-4o* model performed considerably worse with a Recall of 0.076 (-41.1%) and Hits@10 at 0.076 (-41.1%). This is surprising because similar results are not reflected in the experiments of other approaches. In addition, the open-source models tested failed almost completely on the task as their scores are near zero. We assume that this can be attributed to the requirements for structured LLM outputs that the FiDeLiS approach has, which these models struggled to produce consistently. Based solely on the results of this experiment, ***gpt-O3-mini*** is the best choice for the final FiDeLiS configuration.

Embedding Model: The choice of the embedding model also influenced the results of FiDeLiS. The baseline *mxbai-embed-large* model achieved the score 0.129 for both Recall and Hits@10. The *text-embedding-3-large* model yielded a slightly better but almost negligible performance, with a Recall of 0.137 (+6.2%) and Hits@10 of 0.136 (+5.4%). In contrast, *granite-embedding* performed significantly worse with a score of 0.083 (-35.7%) on both metrics. Based on achieving the highest scores in the comparison, we selected ***text-embedding-3-large*** for the final configuration.

Parameter	Config	Recall	Hits@10
Large Language Model	gpt-4o-mini	0.129	0.129
	gpt-4o	0.076 (-41.1%)	0.076 (-41.1%)
	gpt-O3-mini	**0.136 (+5.4%)**	**0.136 (+5.4%)**
	qwen2.5:14b	0.004 (-96.9%)	0.004 (-96.9%)
	llama3.1	0.011 (-91.5%)	0.000 (-100.0%)
Embedding Model	mxbai-embed-large	**0.129**	**0.129**
	text-embedding-3-large	0.137 (+6.2%)	0.136 (+5.4%)
	granite-embedding	0.083 (-35.7%)	0.083 (-35.7%)
Top k	10	**0.129**	**0.129**
	20	0.083 (-35.7%)	0.072 (-44.2%)
	30	0.068 (-47.3%)	0.053 (-58.9%)
Top n	10	**0.129**	**0.129**
	20	0.118 (-8.5%)	0.117 (-9.3%)
	30	0.110 (-14.7%)	0.110 (-14.7%)
Alpha	0.1	0.095 (-26.4%)	0.095 (-26.4%)
	0.3	**0.129**	**0.129**
	0.6	0.110 (-14.7%)	0.110 (-14.7%)
Do Reranking	False	**0.129**	**0.129**
	True	0.118 (-8.5%)	0.117 (-9.3%)
Do Question Augmentation	False	0.129	0.129
	True	0.072 (-44.2%)	0.072 (-44.2%)

Table 9.9.: The results for the parameter selection process of FiDeLiS. The base configuration parameter is underlined, and the highest metric score per parameter is **bold**.

Top k: The baseline value of 10 resulted in a Recall and Hits@10 of 0.129. Increasing the parameter to 20 decreased Recall by 35.7% and Hits@10 by 44.2%, while setting the parameter to 30 resulted in an even larger drop of 47.3% for Recall and 58.9% for Hits@10. This indicates that increasing the width worsened the results. Therefore, we selected the value of **10** for the final configuration.

Top n: For the top-n parameter, the baseline value of 10 achieved a Recall and Hits@10 of 0.129. Increasing the value to 20 resulted in a Recall of 0.118 (-8.5%) and Hits@10 of 0.117 (-9.3%), lowering the performance. Increasing the top-n parameter to 30 decreased performance again by -14.7%. Although the absolute differences are small, given the low overall scores, the trend indicates that the baseline value performed the best. Therefore, we set the parameter value to **10** in the final configuration.

Alpha: The authors of the paper suggest using an alpha score of 0.3. It achieved a Recall and Hits@10 of 0.129. Lowering the value to 0.1 decreased the scores to 0.095 (-26.4%) and

Parameter	Value
`Top k`	10
`Top n`	10
`Alpha`	0.3
`LLM`	gpt-o3-mini
`Embedding Model`	text-embedding-3-large
`Do Question Augmentation`	False
`Do Reranking`	False

Table 9.10.: The final configuration used for subsequent experiments for the FiDeLiS KGQA baseline approach.

increasing the value to 0.6 reduced the scores to 0.110 (-14.7%). As a result of this data, we set the value to **0.3** in the final configuration.

Do Reranking: Not using the reranking step after the retrieval yielded Recall and Hits@10 scores of 0.129. Enabling the reranking resulted in lower performance for both Recall (-8.5%) and Hits@10 (-9.3%). As the reranking process negatively affected the results, we **disabled** it for the final configuration of FiDeLiS.

Do Question Augmentation: Similar to the reranking step, enabling question augmentation proved detrimental to the performance. Compared to the baseline without augmentation, which has both scores at 0.129, enabling it caused a substantial drop in both Recall and Hits@10 to 0.072 (-44.2%). Given this significant negative impact, we **disabled** the augmentation for the final configuration.

9.5. Parameter Selection for Mindmap

In the sections that follow, we initially outline the base configuration and the range of parameters explored during the parameter selection phase for the Mindmap KGQA baseline approach. Subsequently, we analyze the results of the test runs and detail the parameter values chosen for the final setup utilized in our subsequent experiments.

9.5.1. Base Configuration and Parameter Ranges

The parameter values for the base configuration, together with the ranges of parameters tested for Mindmap, are shown in Table 9.11. In what follows, we provide a concise overview of each parameter individually.

Parameter	Parameter Space
`Final Paths To Keep`	10, 20, 30
`Shortest Paths To Keep`	10, 20, 30
`Neighbors to Keep`	10, 20, 30
`Do Question Augmentation`	False, True
`Do Reranking`	False, True
`LLM`	gpt-4o-mini, gpt-4o, o3-mini, Qwen2.5-14B, Llama3.1-8B
`Embedding Model`	mxbai-embed-large, text-embedding-3-large, granite-embedding

Table 9.11.: The base configuration (underlined) and parameter space for the Mindmap KGQA approach.

Final Paths To Keep: This parameter determines how many of the computed evidence paths are retained for the final answer. Increasing this value allows for the consideration of more context but also increases cost and may introduce noise. We initially set this value to 10 because this is the maximum number of golden triples requested by the applied KGQA dataset. We then tested the values 20 and 30.

Shortest Paths to Keep: This parameter determines how many candidate shortest paths are retained during the search between entities. A higher value allows to consider more context but also increases cost and complexity. Same as with the previous parameter, the base configuration value has been set to 10 and we tested the values 20 and 30.

Neighbors to Keep: This parameter determines how many one-hop neighbor relationships are included when building the prompt for the LLM. As with the previous parameters, a higher value allows for more context but increases cost and complexity. We set it to 10 by default and increased the values to 20 and 30.

Do Question Augmentation: Same as with the other KGQA approaches, we disabled the question augmentation in the base configuration and assessed its impact on the results during testing.

Do Reranking: Same as with the other KGQA approaches, we disabled the reranking by default and assessed the impact of enabling the reranking in one of the test runs.

LLM: The LLM has several tasks in the Mindmap approach. First, it is responsible for the extraction of entities from the question. Second, it transforms the path information into a natural language description. Lastly, it generates the final answer. The models that were tried out are introduced in Section 9.1.5. In our initial setup, we employed two models, namely *gpt-4o-mini* and *Qwen2.5-14B*, to decrease the runtime of the parameter selection process by executing several configurations simultaneously. Consequently, embedding models and other LLMs were evaluated using the *Qwen2.5-14B* model as the baseline, while all other parameters were tested with *gpt-4o-mini* as the foundation.

Parameter	Config	Recall	Hits@10
Large Language Model	gpt-4o-mini	0.113	**0.029**
	gpt-4o	0.107 (-5.3%)	0.021 (-27.6%)
	gpt-O3-mini	**0.113 (+0.0%)**	0.013 (-55.2%)
	qwen2.5:14b	0.085 (-24.8%)	0.008 (-72.4%)
	llama3.1	0.000 (-100.0%)	0.000 (-100.0%)
Embedding Model	mxbai-embed-large	0.085	0.008
	text-embedding-3-large	**0.118 (+38.8%)**	0.006 (-25.0%)
	granite-embedding	0.083 (-2.4%)	**0.019 (+137.5%)**
Neighbors to Keep	10	**0.113**	**0.029**
	20	0.113 (+0.0%)	0.029 (+0.0%)
	30	0.113 (+0.0%)	0.029 (+0.0%)
Final Paths to Keep	10	**0.113**	**0.029**
	20	0.113 (+0.0%)	0.029 (+0.0%)
	30	0.113 (+0.0%)	0.029 (+0.0%)
Shortest Paths to Keep	10	**0.113**	**0.029**
	20	0.113 (+0.0%)	0.029 (+0.0%)
	30	0.113 (+0.0%)	0.029 (+0.0%)
Do Reranking	False	**0.129**	**0.129**
	True	0.118 (-6.2%)	0.121 (-6.2%)
Do Question Augmentation	False	0.129	0.129
	True	0.129 (±0.0%)	0.129 (±0.0%)

Table 9.12.: Results of the parameter selection process for Mindmap. The base configuration parameter is underlined, and the highest metric score per parameter is **bold**.

Embedding Model: The embedding model determines the model that is used for embedding the entities of the knowledge graph and the question. The models that we used are introduced in Section 9.1.5. We use *mxbai-embed-large* in the base configuration because of its runtime and cost efficiency.

9.5.2. Parameter Selection

The parameter selection results for the Mindmap approach with 16 configurations are detailed in Table 9.12, with the final configuration shown in Table 9.13. Similarly to the FiDeLiS approach, we observed low performance across all configurations, particularly for the Hits@10 metric, which often remains near zero. This indicates that Mindmap significantly struggles with this question-answering task, which we further discuss in Section 10.3. Given these data, we only reviewed the Recall value as the Hits@10 scores were too low to observe significant differences. Moreover, given the generally low scores, it

is not possible to attribute the results to parameter changes or random variations in the retrieval process. Consequently, our decisions have been made with limited confidence because of these issues.

Large Language Model: The baseline model that was used is *gpt-4o-mini* which scored a Recall of 0.113. The model *gpt-o3-mini* has the same Recall value of 0.113 while *gpt-4o* lost 5.3%. Similarly to FiDeLis, open source models performed poorly. The *Llama3.1* model failed entirely and the *qwen2.5:14b* model lost 24.8% in Recall. Given that *gpt-4o-mini* and ***gpt-o3-mini*** achieved the same Recall, we decided on the latter for the final configuration because the general trend for the other approaches also tends towards this model.

Embedding Model: The baseline model *mxbai-embed-large* achieved a Recall of 0.085, while the *text-embedding-3-large* boosted the Recall value to 0.118 (+38.8%). In contrast, *granite-embedding* caused a minor dip in Recall (-2.4%). Given these data, the final configuration used the ***text-embedding-3-large*** model.

Path and Neighbor Parameters: The tests for the parameters `Final Paths to Keep`, `Shortest Paths to Keep`, and `Neighbor Paths to Keep` were run with a baseline value of 10. In all three cases, increasing the value to 20 or 30 had no measurable effect. Since increasing the limits did not provide any benefit, we retained the base configuration values for the final configuration.

Do Reranking: Disabling reranking resulted in a Recall and Hits@10 value of 0.129. Enabling reranking should theoretically improve the Hits@10 score if the approach does not already rank the values by their relevance. However, enabling the reranking resulted in lower scores for both metrics with a Recall of 0.118 (-6.2%) and Hits@10 of 0.121 (-6.2%). Consequently, since reranking did not improve the performance, we **disabled** it for the final configuration.

Do Question Augmentation: Disabling the augmentation yielded a Recall and Hits@10 value 0.129. Enabling augmentation did not change the results. Because the results of other approaches suggest that the augmentation can negatively affect performance, we decided to **disable** the augmentation.

9.6. Threats to Validity

In this section, we discuss threats to validity specific to the parameter selection process. The goal of the parameter selection process was to find a near-optimal value for each parameter of the KGQA approaches, capable of achieving high overall retrieval performance on our KGQA dataset.

There is a risk that we did not achieve this goal for every parameter we tested. This is due to the limited number of questions used and the limited parameter ranges tested. We were

Parameter	Value
`Final Paths to Keep`	10
`Shortest Paths to Keep`	10
`Neighbor Paths to Keep`	10
`LLM`	gpt-o3-mini
`Embedding Model`	text-embedding-3-large
`Do Question Augmentation`	False
`Do Reranking`	False

Table 9.13.: The final configuration used for subsequent experiments for the Mindmap KGQA approach.

required to impose these constraints to limit the overall execution time of the selection process to stay within the scope of this master thesis. Therefore, our requirement was not to achieve an optimum but to get as close to it as possible, given the constraints. We minimized this risk by carefully considering parameter ranges to ensure that each value tested was both useful and plausible.

Furthermore, we recognize a risk regarding the HubLink approach concerning model selection. This is because, during our development, the prompts were tested and optimized on the *gpt-4o-mini* model. Consequently, there could be a bias towards OpenAI models. Moreover, we were unable to run larger open-source models as we were restricted by hardware requirements.

Additionally, the results exhibit some level of stochasticity, which means that for the same question, the contexts retrieved by the retriever can vary. This can be attributed to two phenomena. First, since we are working with LLMs, they exhibit a certain degree of randomness in their outputs. However, to keep this randomness low, we set the temperature parameter to zero, which reduces the risk of varying outputs. Second, some retrievers work with embedding models, which also have a certain degree of randomness when transforming text into vectors. Finally, the vectors are stored in the Chroma[3] database, which uses the HNSW [152] indexing algorithm that is inherently non-deterministic to a certain degree (see Section 2.5). We further observed that each time the database starts, the index is reinitialized, which can cause minor changes to vectors that are very close to each other in the ANN search compared to previous runs. Despite this, our analysis remains valid, as the randomness induces negligible variations that we do not see as significant enough to impact the overall findings.

[3] `https://www.trychroma.com/` [last accessed 28.03.2025]

10. Evaluation

This chapter presents our evaluation of HubLink, which constitutes contribution **C1** of this thesis. We conducted the evaluation following the GQM plan that we outlined in Section 8.4.3. To establish a comparative baseline, we incorporated five KGQA approaches sourced from existing literature for which we detail the selection process in Section 7.4. All KGQA approaches, including the baselines, were executed using their final configurations, which we determined through the parameter selection process described in Chapter 9. Particularly for our HubLink approach, the chosen configuration along with three additional variations have been evaluated:

1. *HubLink (T)*: This configuration has been selected through our parameter selection process (see Table 9.4) and utilizes the *graph traversal* strategy of HubLink.
2. *HubLink (D)*: This variant employs the same configuration parameters as HubLink (T) but instead uses the *direct retrieval* strategy.
3. *HubLink (F)*: We designed this configuration for reduced runtime, employing the *direct retrieval* strategy, and limiting the *number of hubs* to 10 per question.
4. *HubLink (O)*: This variant shares its parameters with HubLink (T) but utilizes the *mxbai-embed-large* embedding model and the *Qwen2.5-14B* LLM.

The first variant HubLink (T) is expected to achieve the best performance across the variants, as the parameters have been chosen by the parameter selection process. We introduced the other variants to better understand the performance characteristics of HubLink under different operational conditions. Specifically, HubLink (D) allows us to compare the efficacy of the *direct retrieval* strategy against the *graph traversal* strategy used in HubLink (T), which are explained in Section 4.1.2. HubLink (F) aims to provide an economical version optimized for rapid execution, which is a crucial factor for practical usability. Finally, HubLink (O) enables us to assess the performance of our approach when using open-source models. However, the open-source variant is not expected to yield competitive performance compared to the OpenAI models. This is because we were only able to run smaller models due to hardware constraints. Still, including this configuration makes it possible to understand the difference in performance when using smaller models.

Furthermore, our evaluation incorporated four distinct graph variants, which we introduce in Section 7.3.3. However, because of the high cost involved, we were unable to test all the experiments for each of the variants. Consequently, we chose the graph variant **GV1** for the experiments in which the use of multiple variants was not necessary for the reasons provided in Section 9.1.3.

In the sections that follow, we first discuss the evaluation results related to retrieval performance (**ReT1**) in Section 10.1. Subsequently, Section 10.2 analyzes the evaluation results concerning answer generation performance (**GeT1** and **GeT2**). We then conclude our evaluation with a comprehensive discussion of the findings in Section 10.3, before finally addressing threats to validity in Section 10.4.

10.1. Evaluating Retrieval Quality

In this section, we present our evaluation results addressing retrieval target **ReT1**. We begin by analyzing the retrieval performance of HubLink in comparison to the baseline approaches, with a focus on the accuracy and relevance of the retrieved contexts. Subsequently, we examine the impact that the chosen retrieval operation has on overall retrieval performance. Following this, we investigate the applicability and performance of HubLink in various scholarly literature search use cases. We then analyze the influence of type information, when present in the input question, on retrieval outcomes. Next, we assess the robustness of HubLink to structural and lexical variations within the graph. Furthermore, we analyze the relationship between retrieval performance metrics, runtime, and LLM token efficiency. Finally, we evaluate the environmental impact of HubLink relative to the baseline KGQA approaches.

10.1.1. Improvement of Retrieval Accuracy and Relevance

To effectively assist researchers in scholarly literature searches, a KGQA approach must be capable of extracting a broad range of relevant information from the RKG that matches a given question. In the following, we analyze the performance of our proposed HubLink approach in comparison to established KGQA approaches, specifically focusing on the accuracy and relevance of the retrieved contexts. Here, accuracy refers to the extent to which the approach is capable of fetching the desired triples from the graph, while relevance pertains to the degree to which the retrieved triples correspond to the golden triples.

The evaluation results in Table 10.1 demonstrate that HubLink substantially improves the accuracy and relevance of retrieved triples compared to established KGQA baseline methods. In the following, we first discuss the results regarding the different HubLink variants before we focus on the comparison of the baseline approaches.

10.1.1.1. Analyzing HubLink Variants

The HubLink variant (T), which employs the graph traversal strategy, has achieved the highest scores among all variants. With regard to variant (D), which uses the direct retrieval strategy, we observe a decrease in performance. Specifically, the F1 score for HubLink (D) decreases by approximately 15.55% (from 0.328 to 0.277), and MAP@10 shows a reduction of approximately 13.38% (from 0.299 to 0.259). The MRR@10 metric experiences a smaller

Approach	Recall	Precision	F1	Hits@10	MAP@10	MRR@10	EM@10
HubLink (T)	**0.754**	0.246	0.328	**0.512**	**0.299**	0.502	**0.298**
HubLink (D)	0.709	0.221	0.277	0.436	0.259	0.486	0.273
HubLink (F)	0.649	**0.278**	**0.344**	0.451	0.267	0.473	0.290
HubLink (O)	0.559	0.144	0.188	0.408	0.272	**0.526**	0.222
DiFaR	0.352	0.011	0.022	0.208	0.151	0.297	0.104
Mindmap	0.119	0.030	0.045	0.015	0.002	0.013	0.007
FiDeLiS	0.093	0.053	0.063	0.093	0.063	0.103	0.053

Table 10.1.: Comparative overall retrieval performance of HubLink and baseline KGQA approaches on graph variant **GV1** of the ORKG. All presented metrics are macro-averaged.

decrease of approximately 3.19% (from 0.502 to 0.486), while EM@10 drops by approximately 8.39% (from 0.298 to 0.273). In particular, the Hits@10 score sees a substantial decrease of approximately 14.84% (from 0.512 to 0.436) for HubLink (D) compared to HubLink (T). These results suggest that, for our data, the graph traversal strategy is more effective. Although the direct retrieval strategy (HubLink (D)) offers the operational advantage of not requiring a predefined topic entity, this convenience comes at the cost of reduced retrieval efficacy. While we hypothesize that the size of the tested RKG may not fully represent the differences between the strategies, the observed performance degradation with direct retrieval could already indicate potential challenges that could be amplified on larger and more complex graph structures. This aspect warrants further investigation in future work.

The HubLink (F) variant, designed for reduced runtime, presents a notable trade-off. Compared to HubLink (T), it shows a lower Recall (0.649 versus 0.754) and Hits@10 (0.451 versus 0.512). However, HubLink (F) achieves the highest Precision (0.278) and F1 score (0.344) among all evaluated variants, surpassing even HubLink (T) (Precision 0.246, F1 0.328). In our opinion, the improved Precision score is attributed to the reduced number of triples that this variant retrieves. This is because, by retrieving fewer triples overall, the proportion of truly relevant triples among those retrieved can be higher, thus increasing Precision. The results also underscore the significant role that the number of considered hubs plays in retrieval performance.

The open-source variant HubLink (O) generally shows lower performance in most metrics compared to the other HubLink configurations, as it records the lowest metric scores across all variants. This outcome is consistent with previous observations from the parameter selection process (see Section 9.2). This suggests that the capability of the applied LLM has a substantial impact on the retrieval performance. An interesting exception is MRR@10, where HubLink (O) achieves the highest score of all variants. This indicates that when HubLink (O) does identify a correct triple within the top 10 results, that triple is often ranked high. However, the lower Hits@10 implies that the model is less frequently successful in placing a correct triple within the top 10.

10.1.1.2. Analyzing against Baselines

HubLink (T) achieved a Recall of 0.754, representing a 114% improvement over the next best KGQA approach (DiFaR), which reached a Recall of 0.352. In contrast, Mindmap and FiDeLiS retrieved only approximately 10% of the expected triples from the graph. This result indicates that the baseline approaches struggled significantly with the task. In particular, both DiFaR and HubLink utilize dense vector retrieval mechanisms, while Mindmap and FiDeLiS rely on subgraph construction and stepwise reasoning, respectively. These findings suggest that in the context of our evaluation, embedding-based approaches offer a clear advantage in retrieving a larger proportion of relevant triples.

Although Precision values were generally lower across all models, HubLink (T) again outperformed baselines with a Precision of 0.246, highlighting its relatively greater ability to return correct triples. However, the absolute score is rather low, which indicates that a large number of irrelevant triples is also retrieved. However, as discussed in Section 9.1.2, these triples may still contribute positively to answer generation.

In evaluating the effectiveness of the ranking, HubLink (T) achieved a Hits@10 score of 0.512, which is more than twice as high as the next best baseline. However, this metric also reveals that the most relevant triples did not always appear at the top, suggesting weaknesses in overall ranking performance. The other ranking metrics further clarify this pattern. The MAP@10 score of 0.299 and the MRR@10 score of 0.502 indicate that while HubLink generally ranks relevant triples higher than baselines, its ability to consistently prioritize them in the topmost positions remains limited. Nevertheless, these scores are significantly higher than those of other methods, confirming a substantial advancement in contextual relevance and ranking quality over baseline methods.

10.1.1.3. Discussion on Overall Retrieval Performance

Our analysis of HubLink variants and the comparison to baseline methods reveals significant advancements in retrieval accuracy and relevance for scholarly literature search. We determined that the graph traversal strategy yields superior performance over the direct retrieval strategy. Furthermore, we observed that increasing the number of hubs during retrieval enhances Recall. However, this enhancement corresponds to a reduction in Precision and ranking performance. These outcomes suggest that, while more hubs allow for a broader retrieval scope, thereby capturing a larger set of potentially relevant items, the specificity to the query context diminishes. Moreover, the results for HubLink (O) affirm the critical role that the choice of the LLM and the underlying embedding model plays in the retrieval performance of HubLink. Additionally, the absolute Precision and ranking scores indicate a current limitation in the ability of HubLink to accurately assess the relevance of triples.

When we contrast HubLink with established baseline approaches, the advantages of our method become particularly clear. All HubLink variants demonstrate a marked improvement

across all evaluated metrics. We generally observe low performance from the baseline approaches, suggesting their limited applicability to scholarly literature searches. Interestingly, the data indicate that embedding-based methods are superior for this task.

> **Answer to Q1:** *Our HubLink retrieval approach significantly improves retrieval performance with respect to relevance and Precision compared to established baseline KGQA methods in the scholarly literature search setting. The approach more than doubles the Recall of the next best baseline, indicating a substantially better retrieval of relevant triples. Precision is also markedly higher, suggesting an improved assessment of relevance during retrieval. However, the absolute Precision and ranking performance of HubLink highlights a weakness and a potential need for improvement. Nevertheless, these results collectively demonstrate that HubLink offers a notable improvement in the retrieval of contextually relevant triples.*

10.1.2. Impact of Operation Complexity

To effectively address a wide array of scholarly questions characterized by varying semantic and logical complexity, a KGQA approach must be able to handle diverse reasoning operations. In the following, we analyze the performance of HubLink across different operations, which are provided by our KGQA retrieval taxonomy (see Chapter 6).

The results of the experiment are presented in Table 10.2, which shows the retrieval performance of each of the four HubLink variants for eight distinct operations.

For *Basic* operations, which entail the retrieval of a single triple without further processing, all HubLink variants demonstrate their highest Recall and Hits@10 scores. These findings suggest that the approach shows the best performance in retrieving triples for simple, fact-based questions solvable through single triple lookups.

Furthermore, although HubLink (T) and HubLink (O) have achieved the highest F1 scores with the *Basic* operation questions, the same is not true for the other variants. HubLink (D) and HubLink (F) achieved the highest F1 scores with *Comparative* and *Relationship* questions. Moreover, we observe that Precision and F1 scores are generally lower than Recall across all operations, suggesting inherent difficulties for the retriever in precisely identifying relevant information. This is particularly evident in the case of *Negation* and *Superlative* operations, which consistently yield lower Precision and F1 scores across all variants when compared to other operations. This pattern indicates that the retriever faces challenges in accurately identifying relevant information when these logical constructs are involved. The Hits@10 metric further confirms this trend, as it also shows lower scores for these operations.

An examination of the MAP@10, MRR@10, and EM@10 metrics does not reveal a clear, overarching trend in performance across the different operations that is consistent for all HubLink variants. However, we observe a minor trend where *negation* and *superlative* operations tend to yield lower scores.

Retrieval Operation	Recall	Precision	F1	Hits@10	MAP@10	MRR@10	EM@10
HubLink (T)							
basic	**0.917**	**0.382**	**0.480**	**0.917**	**0.445**	0.490	**0.389**
aggregation	0.810	0.209	0.285	0.497	0.225	0.347	0.240
counting	0.840	0.275	0.372	0.644	0.357	0.526	0.340
ranking	0.817	0.321	0.414	0.561	0.360	**0.576**	0.363
comparative	0.742	0.262	0.366	0.456	0.320	0.560	0.296
relationship	0.628	0.254	0.314	0.410	0.298	0.528	0.331
negation	0.584	0.072	0.122	0.244	0.125	0.419	0.144
superlative	0.656	0.129	0.193	0.319	0.207	0.540	0.237
HubLink (D)							
basic	**0.861**	0.217	0.276	**0.611**	0.297	0.332	0.228
aggregation	0.730	0.166	0.217	0.388	0.188	0.365	0.200
counting	0.723	0.218	0.293	0.481	0.287	0.410	0.253
ranking	0.659	0.221	0.278	0.428	0.278	0.494	0.269
comparative	0.701	0.314	**0.376**	0.444	0.287	0.537	0.339
relationship	0.689	**0.347**	**0.376**	0.456	**0.314**	0.627	**0.411**
negation	0.639	0.065	0.118	0.325	0.169	0.534	0.200
superlative	0.690	0.133	0.204	0.332	0.229	**0.635**	0.244
HubLink (F)							
basic	**0.806**	0.279	0.338	**0.694**	**0.364**	0.392	0.287
aggregation	0.652	0.215	0.277	0.427	0.172	0.267	0.235
counting	0.779	0.273	0.376	0.477	0.301	0.561	0.273
ranking	0.630	0.236	0.320	0.404	0.224	0.458	0.235
comparative	0.617	0.366	0.428	0.504	0.355	0.549	0.366
relationship	0.610	**0.420**	**0.443**	0.456	0.310	0.541	**0.428**
negation	0.509	0.148	0.216	0.311	0.177	0.487	0.211
superlative	0.575	0.251	0.315	0.325	0.232	**0.572**	0.263
HubLink (O)							
basic	**0.806**	**0.280**	**0.345**	**0.806**	**0.590**	0.617	0.297
aggregation	0.638	0.123	0.164	0.426	0.250	0.391	0.204
counting	0.710	0.170	0.249	0.600	0.404	**0.714**	0.273
ranking	0.611	0.128	0.188	0.394	0.256	0.514	0.234
comparative	0.387	0.075	0.118	0.285	0.179	0.443	0.150
relationship	0.465	0.265	0.274	0.328	0.259	0.653	**0.328**
negation	0.499	0.039	0.067	0.232	0.121	0.475	0.144
superlative	0.322	0.031	0.055	0.149	0.084	0.382	0.106

Table 10.2.: Impact of the retrieval operation on the performance of the HubLink approach. The results are based on graph variant **GV1** and all metrics have been macro-averaged. The results for the baseline approaches are provided in Appendix A.3.1.

Answer to Q2: *The results indicate that the highest Recall is achieved with basic operation questions, with a noticeable performance drop for questions demanding more complex reasoning operations. Furthermore, the results suggest a general limitation in assessing relevance, as the retriever consistently retrieves more contexts than asked for and struggles to differentiate effectively between relevant and irrelevant contexts. This difficulty is particularly pronounced for negation and superlative operations.*

10.1.3. Applicability to Different Scholarly Literature Search Use Cases

Table 10.3 presents the evaluation results for the HubLink approach across six predefined scholarly literature search use cases. These use cases, as detailed in Section 8.3.1, are distinguished by their input condition type (metadata, content, or both) and expected answer type (metadata or content). The subsequent discussion analyzes the performance across the four different HubLink variants.

Concerning Recall performance, use cases involving metadata conditions in the query demonstrate superior results. Specifically, Use Case 1 (metadata input, metadata output) and Use Case 2 (metadata input, content output) generally have the highest Recall scores across all HubLink variants. Use Case 5 (mixed input, content output) and Use Case 6 (mixed input, metadata output), which also include metadata conditions, tend to follow closely in Recall performance. This pattern suggests that the approach encounters greater challenges with query conditions based solely on *content*, as Use Case 3 (content input, metadata output) and Use Case 4 (content input, content output) consistently show lower Recall values.

We further observe that the second, fourth, and fifth use cases, which require content-type answers, tend to exhibit lower Precision scores across all HubLink variants. This suggests that the approach is less effective in accurately identifying relevant information when the expected answer type is content. Regarding the ranking metrics (MAP@10, MRR@10, EM@10), Use Case 1 consistently demonstrates the strongest performance across all HubLink variants for most of these metrics. However, for the other five use cases, no consistently discernible pattern emerges that would suggest a general superiority or inferiority of any specific use case across all variants or ranking metrics.

Overall, the results suggest that optimal performance is most frequently achieved in use cases that involve metadata-based query conditions or require metadata as output. This observation aligns with current scholarly research practice, where researchers typically search by titles or keywords. Nevertheless, the data indicate that the Recall performance of HubLink for queries requiring content-specific information is also relatively high. This suggests that HubLink has the potential to transform current research workflows from metadata-based to content-based searches. However, our findings also reveal limitations in effective ranking and filtering, particularly when dealing with content-based query conditions or answers, as well as mixed condition scenarios. This indicates that the integration of subsequent filtering and reranking mechanisms could be beneficial to mitigate these limitations.

Use Case	Recall	Precision	F1	Hits@10	MAP@10	MRR@10	EM@10
HubLink (T)							
1	0.800	**0.507**	**0.575**	**0.767**	**0.557**	**0.644**	**0.552**
2	**0.848**	0.252	0.364	0.729	0.301	0.341	0.281
3	0.768	0.252	0.343	0.507	0.268	0.543	0.287
4	0.663	0.198	0.277	0.395	0.266	0.561	0.255
5	0.702	0.122	0.186	0.350	0.184	0.408	0.213
6	0.779	0.206	0.286	0.428	0.278	0.512	0.257
HubLink (D)							
1	**0.791**	**0.450**	**0.510**	**0.745**	**0.495**	0.556	**0.489**
2	0.715	0.073	0.127	0.410	0.155	0.281	0.111
3	0.675	0.265	0.332	0.444	0.234	0.459	0.298
4	0.543	0.195	0.225	0.302	0.184	0.444	0.234
5	0.756	0.144	0.191	0.317	0.183	0.481	0.236
6	0.790	0.213	0.295	0.463	0.341	**0.691**	0.284
HubLink (F)							
1	**0.770**	**0.524**	**0.591**	**0.758**	**0.499**	0.538	**0.533**
2	0.674	0.150	0.228	0.438	0.202	0.272	0.158
3	0.611	0.262	0.313	0.347	0.162	0.385	0.240
4	0.524	0.268	0.330	0.390	0.254	0.503	0.291
5	0.675	0.234	0.308	0.370	0.247	**0.597**	0.253
6	0.676	0.271	0.336	0.489	0.293	0.534	0.306
HubLink (O)							
1	0.689	**0.370**	**0.436**	0.667	**0.581**	**0.719**	**0.395**
2	**0.776**	0.195	0.284	**0.690**	0.297	0.380	0.250
3	0.531	0.152	0.162	0.374	0.275	0.512	0.254
4	0.357	0.047	0.068	0.220	0.156	0.473	0.148
5	0.463	0.044	0.077	0.251	0.150	0.484	0.135
6	0.609	0.112	0.176	0.360	0.243	0.620	0.195

Table 10.3.: Assessment of different scholarly use cases on the retrieval performance of HubLink. The results are based on graph variant **GV1** and all metrics have been macro-averaged. The use cases are introduced in Section 8.3.1. The results for the baseline approaches are provided in Appendix A.3.2.

Answer to Q3: *The Recall performance of the proposed HubLink approach is generally robust across most use cases. However, the overall performance for Precision and ranking effectiveness tends to decline in scenarios involving content-based question conditions or when content-type answers are required. The approach often includes a higher proportion of irrelevant information and faces challenges in ranking the most relevant triples at the top positions for content-focused use cases.*

Semi-Typed	Recall	Precision	F1	Hits@10	MAP@10	MRR@10	EM@10
			HubLink (T)				
True	**0.763**	**0.258**	**0.352**	**0.567**	**0.328**	**0.512**	**0.323**
False	0.747	0.233	0.302	0.456	0.268	0.488	0.273
			HubLink (D)				
True	**0.730**	**0.229**	**0.302**	**0.461**	**0.272**	**0.493**	**0.284**
False	0.691	0.212	0.251	0.411	0.247	0.485	0.263
			HubLink (F)				
True	**0.680**	**0.321**	**0.394**	**0.498**	**0.318**	**0.535**	**0.334**
False	0.622	0.237	0.296	0.407	0.218	0.415	0.248
			HubLink (O)				
True	0.377	0.258	0.495	0.110	0.537	0.156	0.196
False	**0.441**	**0.287**	**0.565**	**0.179**	**0.583**	**0.222**	**0.251**

Table 10.4.: The impact of questions that add information about the condition types compared to those that do not. The results are based on graph variant **GV1** and all metrics have been macro-averaged. The results for the baseline approaches are provided in Appendix A.3.3.

10.1.4. Impact of Type Information in the Question

In Table 10.4, the performance of four different HubLink variants is presented, comparing questions that include semi-typed information with those that do not. Such type annotations could assist the retriever in disambiguating the roles of entities within the graph and narrowing the candidate search space by filtering semantically irrelevant triples.

The results indicate that semi-typed questions lead to improved performance across all reported metrics for HubLink variants (T), (D), and (F). The magnitude of these improvements is generally modest for variants (T) and (D), while variant (F) exhibits more noticeable gains, particularly in Precision, F1, MAP@10, MRR@10, and EM@10. Conversely, for HubLink variant (O), which uses an open-source LLM and embedding model, the inclusion of type information results in a performance decline across all metrics compared to when type information is absent.

Although a consistent positive trend is observed for variants (T), (D), and (F) and a negative trend is observed for variant (O), the differences in performance, especially for (T) and (D), are relatively small. Overall, while the presence of type information appears to exhibit a minor to moderate positive effect for the non-open-source variants, its impact is not consistently substantial across all these variants in the current experimental setup. Interestingly, for the open-source variant, the inclusion of type information seems to have a negative effect on retrieval performance.

Answer to Q4: *A minor positive impact on retrieval performance is observed for non-open-source LLMs when explicit type information is included in questions. However, the magnitude of this positive effect is quite small and is not significant. In contrast, the open-source variant shows a negative impact when type information is included.*

10.1.5. Robustness to Structural and Lexical Variability in Graph Schema

In the following section, we analyze the robustness of the HubLink retrieval approach in terms of performance consistency across different graph schemas introduced in Section 7.3.3. Furthermore, we analyze the impact of the number of hops required to reach the relevant triples. In this context, robustness refers to the ability of the retrieval system to maintain consistent performance in terms of accuracy and relevance.

In the subsequent evaluations, we focus on the performance of HubLink (T), as the execution of all HubLink variants on all graph variants would have been too costly. We chose HubLink variant (T) as it consistently shows the highest retrieval performance in our previous evaluations.

10.1.5.1. Analyzing Different Graph Variants

A key characteristic of the proposed HubLink approach and the applied baseline approaches is their schema-agnostic design, which allows the application to various graph schemas without having to change the configuration or implementation of the approach. To explore the practical implications of this flexibility, we evaluated the performance of HubLink and the baseline approaches on four different graph variants (**GV1-GV4**) introduced in Section 7.3.3. The results are presented in Table 10.5 and discussed in the following.

The retrieval performance of HubLink (T) against the baseline approaches for the first graph variant (**GV1**) has already been extensively discussed in Section 10.1.1. In summary, the results for **GV1** indicate that the HubLink approach is capable of effectively retrieving relevant information from the ORKG and achieving high Recall and ranking scores. The baseline approaches, on the other hand, struggled to achieve comparable performance.

For graph variant **GV2**, which also features long paths similar to **GV1** but incorporates semantic grouping in which all information is stored in a single ORKG contribution, we observe minor improvements for HubLink (T). Although the Recall remains similar to **GV1**, the Precision increased by approximately 12%, leading to a higher F1 score. In addition, minor improvements in ranking performance can also be observed, particularly with MAP@10 increasing by approximately 12% and EM@10 by 8%. This suggests that the semantic grouping of information in **GV2** may have a positive impact on the relevance assessment. This positive trend is further supported by FiDeLiS, which also shows improved performance in **GV2** compared to **GV1**. However, for Mindmap and DiFaR, the results indicate a reduction across all metrics, contradicting this assumption.

Graph	Recall	Precision	F1	Hits@10	MAP@10	MRR@10	EM@10
HubLink (T)							
GV1	0.755	0.246	0.327	0.513	0.298	0.500	0.299
GV2	0.759	0.276	0.352	0.518	0.333	0.522	0.324
GV3	**0.812**	0.350	0.423	0.596	0.408	0.650	0.406
GV4	0.804	**0.393**	**0.452**	**0.597**	**0.425**	**0.661**	**0.444**
DiFaR							
GV1	0.352	0.011	0.022	0.207	0.150	0.295	0.104
GV2	0.314	0.009	0.019	0.199	0.142	0.268	0.096
GV3	0.523	**0.017**	**0.035**	0.302	**0.230**	0.442	0.154
GV4	**0.528**	**0.017**	**0.035**	**0.304**	0.228	**0.449**	**0.158**
Mindmap							
GV1	0.119	0.030	0.045	0.015	0.002	0.013	0.007
GV2	0.093	0.025	0.037	0.008	0.001	0.006	0.005
GV3	0.133	0.043	0.061	**0.030**	0.007	0.023	0.015
GV4	**0.127**	**0.044**	**0.062**	**0.030**	**0.010**	**0.036**	**0.018**
FiDeLiS							
GV1	0.092	0.052	0.063	0.092	0.062	0.103	0.053
GV2	0.099	0.055	0.064	0.099	0.065	0.110	0.054
GV3	0.259	0.114	0.139	0.259	0.150	0.240	0.112
GV4	**0.276**	**0.121**	**0.142**	**0.276**	**0.156**	**0.248**	**0.117**

Table 10.5.: Test results for the evaluation of four different graph variants introduced in Section 7.3.3. All metrics have been macro-averaged.

With **GV3**, characterized by shorter paths and distributing information across multiple ORKG contributions, significant improvements were observed across all models. For HubLink (T), most metrics increase by approximately 15 to 25% when compared to **GV2**. However, for the Recall metric, the increase is only minor (approximately 7%), suggesting that the Recall score stays relatively stable. For the baseline approaches, on the other hand, the shorter paths led to substantial improvements in overall retrieval performance. The scores of DiFaR increased by approximately 52 to 89% across all metrics when compared to **GV2**. Similarly, FiDeLiS more than doubled its scores across all metrics, with the Recall and Hits@10 scores increasing by over 160%. The same trend can be observed for the Mindmap retriever, though the absolute scores still remain low. These results suggest that shorter paths are beneficial for the retrieval performance of all approaches, in particular for the baseline approaches.

Finally, the last graph variant **GV4** combines shorter paths with the semantic grouping of information in a single ORKG contribution. Compared to **GV3**, a minor performance increase can be observed in particular for the Precision (by approximately 12%) and EM@10

(by approximately 9%) scores. Similar improvements can be seen for the baseline methods, but the increase is only minor. These results suggest that the performance impact of grouping information in a single ORKG contribution compared to distributing it across multiple contributions is not significant.

In summary, the evaluations reveal consistent patterns across all KGQA approaches examined: shorter path lengths significantly enhance performance. Moreover, the results indicate that the baseline approaches struggle substantially with longer paths, highlighting their limitations in multi-hop reasoning. In contrast, the performance of HubLink (T) remained substantially superior across all variants, demonstrating high adaptability and robust performance regardless of the underlying graph structure variations.

10.1.5.2. Analyzing Number of Hops

The number of hops signifies the maximum count of triples situated between the topic entity and the expected triples. It indicates how deeply the required information is embedded within the graph. In Table 10.6, the outcomes for HubLink and the baseline methods are presented, categorized by the necessary number of hops required to reach the desired triples. These evaluations were conducted using the graph variant **GV1**. The distribution of questions based on the number of hops is detailed in Table 10.7.

For questions requiring only one or two hops representing information located in the immediate vicinity of the topic entity, distinct performance patterns emerge. In the 1-hop scenario, all methods achieve perfect Recall. However, a significant difference is observed in Precision and rank-based metrics, while the baseline methods exhibit substantially lower performance, particularly DiFaR and Mindmap. For those questions that necessitate two hops, the performance advantage of HubLink becomes more evident. The Recall remains very high at 0.937, indicating retrieval of almost all relevant triples. In contrast, FiDeLiS and DiFaR achieve a Recall of 0.667, while Mindmap fails entirely for this hop count. It is evident from these results that the Mindmap approach has considerable issues with our evaluation task, which we discuss in Section 10.3. Concerning Precision at two hops, the results for HubLink (0.792) again significantly surpass those of the baseline methods.

A notable divergence in performance occurs for questions that require three or more hops, corresponding to information deeper within the graph. Baseline methods generally exhibit a substantial degradation in effectiveness as the hop count increases. The performance of FiDeLiS drops sharply beyond two hops, with the Recall score only reaching 0.060 at three hops and becomes negligible or zero at four and six hops, but interestingly reaches a score 0.126 at five hops. Mindmap consistently yields very low Recall values (between 0.115 and 0.136) and near-zero values for all other metrics. DiFaR demonstrates greater resilience to high hop counts compared to FiDeLiS and Mindmap, maintaining Recall values between 0.285 and 0.465. However, even for DiFaR, a decline relative to the 1- and 2-hop scenarios is observed across all metrics. This collective performance decline across the baseline metrics suggests that the methods face significant challenges in effectively identifying relevant

Hop Count	Recall	Precision	F1	Hits@10	MAP@10	MRR@10	EM@10
HubLink (T)							
1	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
2	0.937	0.792	0.818	0.938	0.755	0.750	0.792
3	0.732	0.358	0.446	0.699	0.427	0.509	0.404
4	0.934	0.300	0.446	0.933	0.408	0.450	0.300
5	0.813	0.212	0.313	0.524	0.268	0.459	0.246
6	0.720	0.188	0.262	0.414	0.238	0.494	0.255
DiFaR							
1	**1.000**	0.010	0.010	**1.000**	0.167	0.167	0.100
2	0.667	0.007	0.013	0.333	0.153	0.250	0.050
3	0.380	0.007	0.018	0.311	**0.307**	**0.469**	**0.113**
4	0.300	0.004	0.004	0.000	0.000	0.000	0.000
5	0.465	0.014	**0.028**	0.210	0.149	0.326	0.109
6	0.285	**0.011**	0.022	0.176	0.120	0.262	0.108
Mindmap							
1	**1.000**	**0.070**	**0.130**	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.136	0.031	0.048	0.005	0.001	0.006	0.004
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.132	0.029	0.043	0.013	0.002	0.011	**0.009**
6	0.115	0.033	0.048	**0.019**	**0.003**	**0.017**	0.008
FiDeLiS							
1	**1.000**	**0.500**	**0.670**	**1.000**	**1.000**	**1.000**	**0.857**
2	0.667	0.358	0.440	0.667	0.639	0.667	0.433
3	0.060	0.037	0.045	0.060	0.052	0.062	0.037
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.126	0.068	0.082	0.126	0.072	0.118	0.067
6	0.050	0.031	0.035	0.050	0.021	0.070	0.024

Table 10.6.: Test results for the evaluation of different hops. The results are based on graph variant **GV1** and all metrics have been macro-averaged.

triples deep within the graph. In particular, the performance at 6 hops is based on a high number of questions, making the observed trend especially relevant.

In contrast, the results for HubLink demonstrate considerably greater robustness to increasing hop counts. The Recall score remains relatively high across all evaluated hop distances, consistently exceeding 0.717 and peaking at 1.00 (1-hop) and 0.934 (4-hops). Furthermore, the Recall value of HubLink consistently exceeds the best performance baseline (DiFaR) by a substantial margin, often nearly doubling the Recall value for hop counts of three and

Hops	Value
1	1
2	6
3	24
4	5
5	33
6	101

Table 10.7.: The distribution of hops for graph variant **GV1**.

more. Similarly, the Precision and rank-based metrics for HubLink remain significantly higher than those of the baselines.

However, the results indicate that the effectiveness of HubLink is influenced by the hop count. We observe a large drop in Precision as the number of hops increases. This decline likely reflects the inherent challenge of maintaining high Precision as the search space expands with each additional hop. Exploring deeper into the graph retrieves a larger set of candidate triples, possibly making the accurate assessment of relevance for each triple more difficult. Rank-based metrics also show a generally decreasing trend for HubLink as hops increase, although the decline is less steep than for Precision. The Recall metric exhibits the least sensitivity to hop count for HubLink, indicating robustness in finding relevant information even when it is stored deep within the graph.

In summary, the analysis based on hop count reveals that HubLink provides a substantial improvement in retrieval performance compared to the evaluated baseline methods. This advantage is particularly pronounced for questions where the target information is located deeper within the KG, requiring higher hop counts. While baseline methods struggle significantly as the number of hops increases, HubLink maintains comparatively high Recall and superior Precision and ranking performance, demonstrating a greater capability to handle complex queries requiring multi-hop reasoning.

10.1.5.3. Discussion on Robustness

Our analysis reveals that HubLink exhibits substantially greater robustness to different graph variants than the baseline approaches. Our findings suggest that a primary factor that influences performance across all approaches is the path length within the graph. We found that shorter paths generally yield better retrieval results, particularly for Precision and rank-based metrics. Although all approaches benefit from shorter paths, the baseline methods demonstrate a pronounced degradation with longer paths, highlighting their limitations. In contrast, HubLink maintains considerably more stable and superior performance.

Furthermore, the types of entities in the four different applied graph variants differ. Because it was not necessary to change the configurations or implementations of the tested KGQA approaches, this demonstrates their schema-agnostic design.

Approach	Recall	Runtime (s)	LLM Tokens
HubLink (T)	**0.754**	177.470	82365
HubLink (D)	0.709	155.387	79467
HubLink (F)	0.649	80.740	28404
HubLink (O)	0.559	239.593	82839
DiFaR	0.352	**3.442**	**25020**
FiDeLiS	0.092	129.914	102178
Mindmap	0.119	90.186	13986

Table 10.8.: Results of runtime and LLM token consumption per question. The results are based on graph variant **GV1** and all metrics have been macro-averaged.

Moreover, the examination of performance across different hop counts underscores the enhanced multi-hop reasoning capabilities of HubLink. As the required number of hops increases, baseline methods show a significant decline in effectiveness. HubLink, however, sustains high Recall and superior Precision and ranking scores, even when navigating multiple hops. While we observe some decline in the Precision for HubLink with increasing hops, its overall ability to retrieve deeply located information far surpasses that of the baselines. In essence, the results indicate that HubLink adapts more effectively to diverse graph structures and complexities, particularly excelling in scenarios demanding robust multi-hop inference.

> **Answer to Q5:** *Our proposed HubLink approach significantly improves robustness compared to baseline methods. Our evaluations show that, while shorter graph paths generally improve retrieval performance, particularly with regard to relevance-focused metrics, HubLink is more effective across varied structures and excels at multi-hop reasoning, an area in which baseline methods struggle.*

10.1.6. Analysis of Runtime and LLM Token Consumption

The runtime and LLM token consumption for the different HubLink versions and baseline approaches are presented in Table 10.8. These metrics provide insights into the computational efficiency and operational costs associated with each approach.

The comparison between the different HubLink variants reveals distinct trade-offs between Recall, runtime, and token consumption. HubLink (T), which uses the graph traversal strategy, had an average runtime of approximately 177.47 seconds and consumed 82,365 LLM tokens per question. Switching to the direct retrieval strategy, HubLink (D) shows improved efficiency, as the runtime decreases to approximately 155.39 seconds, a reduction of about 12.4% compared to HubLink (T). However, token consumption stays almost the same with only a reduction of approximately 3.5%. Although the runtime is slightly lower, the Recall decreases as well (approximately 5.8%).

HubLink (F) is specifically optimized for speed by using the direct retrieval strategy and limiting the number of hubs to 10. It demonstrates the highest efficiency among the HubLink variants, as it required an average runtime of approximately 80.74 seconds per question. This makes HubLink (F) more than twice as fast as HubLink (T). Furthermore, this speed improvement is coupled with a significant decrease in resource usage since only 28,404 LLM tokens are needed. This token count is approximately 65.5% fewer than HubLink (T). However, these efficiency gains result in a lower Recall of 0.649, a relative decrease of approximately 13.8% compared to HubLink (T) and about 8.5% compared to HubLink (D).

Conversely, the open-source version of HubLink (O) exhibits the longest average runtime at approximately 239.59 seconds. Its token consumption (82,839) is nearly identical to that of HubLink (T), an expected outcome given their shared configuration parameters despite differing underlying models. This high resource usage for HubLink (O) is further associated with the lowest Recall (0.559) among the HubLink variants.

Compared with baseline approaches, DiFaR stands out with an exceptional runtime efficiency, processing each question in approximately 3.44 seconds on average. However, the LLM token requirement of 25,020 is only marginally lower than that of HubLink (F) (28,404), indicating comparable operational costs between these two methods despite the vast difference in execution speed. Among the other baselines, Mindmap requires approximately 90.19 seconds per question, slightly slower than HubLink (F), while FiDeLiS takes approximately 129.91 seconds. In terms of token consumption, Mindmap is the most cost-effective approach, using only about 13,986 tokens per question. FiDeLiS, in contrast, incurs the highest cost, requiring over 102,178 tokens, substantially more than any other evaluated method.

In summary, the results demonstrate that all HubLink variants achieve considerably higher Recall than the baseline methods. While DiFaR offers the lowest runtime by a significant margin, its Recall is substantially lower. HubLink (F) emerges as a balanced configuration, providing Recall performance nearly double that of DiFaR (0.649 vs. 0.352) with a runtime that, although higher than DiFaR, remains considerably faster than HubLink (T) or (O). Furthermore, the token consumption, and therefore the estimated operational cost, of HubLink (F) is comparable to DiFaR. Although Mindmap offers the lowest token cost and FiDeLiS incurs the highest, both approaches exhibit significantly lower Recall compared to HubLink. Therefore, HubLink (F) presents a convincing trade-off, delivering strong Recall performance with manageable runtime and reasonable computational expense when compared to the baselines.

> **Answer to Q6:** *The proposed HubLink approach achieves significantly higher Recall compared to baseline KGQA methods, with HubLink (F) balancing Recall, runtime efficiency, and operational token costs. While baseline methods like DiFaR offer superior runtime, they provide considerably lower Recall. HubLink (F) thus presents a favorable trade-off, offering strong retrieval performance alongside manageable computational costs.*

Approach	Recall	CE (CO_2)	CE_{rel}	Δ_{CE}	n(CE)	n(CE_{rel})
HubLink (T)	**0.753905**	0.004187	0.005554	0.482545	0.550789	0.837496
HubLink (D)	0.709053	0.003664	0.005168	0.514019	0.608081	0.849320
HubLink (F)	0.649467	0.001907	0.002936	0.892193	0.800543	0.917609
HubLink (O)	0.559290	0.009216	0.016478	0.154695	min	0.503199
DiFaR	0.352012	**0.000086**	**0.000244**	**9.233593**	max	max
FiDeLiS	0.092840	0.003056	0.032921	base	0.674635	min
Mindmap	0.119467	0.002130	0.017828	0.038211	0.776120	0.461889

Table 10.9.: Results of environmental sustainability analysis. The results are based on graph variant **GV1** and all metrics have been macro-averaged.

10.1.7. Environmental Sustainability Impact

Traditional performance metrics like Recall do not capture the environmental footprint associated with computational methods. Responding to appeals for the incorporation of such factors [78], we evaluate the environmental sustainability of the approaches by analyzing their Carbon Emissions (CE) relative to their Recall performance. The results are based on the graph variant **GV1** and are presented in Table 10.9.

Similar to the previous runtime and cost analysis, we can see distinct sustainability profiles of the HubLink variants relative to their Recall capabilities. Although HubLink (T) achieves the highest Recall (0.753905), it incurs the highest absolute carbon emissions ($CE = 0.004187$ CO_2) relative to HubLink variants (D) and (F). Only HubLink (O) has higher absolute emissions ($CE = 0.009216$ CO_2) but, as indicated below, this is due to measurement limitations. Compared to the baseline approaches, the absolute carbon emissions of DiFaR are significantly lower ($CE = 0.000086$ CO_2), making it the most environmentally efficient approach across all our tested approaches. The consumption of FiDeLiS is similar to that of HubLink (D), and the consumption of Mindmap is similar to that of HubLink (F).

Looking at the relative carbon emissions (CE_{rel}), which is the ratio of carbon emissions to Recall, we observe that DiFaR is the most efficient approach with a relative carbon emission of 0.000244. This is followed by HubLink (F), with a relative carbon emission of 0.002936, which is approximately 12 times higher than DiFaR. The relative carbon emissions of HubLink (D) and (T) are 0.005168 and 0.005554, respectively, indicating that the direct retrieval strategy in HubLink (D) is more efficient than the graph traversal strategy in HubLink (T). The open-source variant, HubLink (O), has the highest relative carbon emissions at 0.016478, making it the least efficient option. While the relative carbon emissions of Mindmap (0.017828) are similar to HubLink (O), its Recall is significantly lower, indicating a less favorable sustainability profile. FiDeLiS has the highest relative carbon emissions (0.032921) among all approaches, making it the least efficient in terms of environmental impact and Recall performance. The same trends are observed for the delta values (Δ_{CE}).

In conclusion, the evaluation shows a trade-off between Recall performance and environmental sustainability. DiFaR provides the most sustainable option, characterized by very

low absolute and relative environmental impacts, but achieves considerably lower Recall than the HubLink approaches. In contrast, the fast HubLink variant (F) almost doubles the Recall compared to DiFaR, but incurs a disproportionately large increase in environmental cost per unit of Recall. Therefore, selecting an appropriate method requires careful consideration of the priority assigned to maximizing Recall versus minimizing environmental footprint. While HubLink (F) represents an optimized balance, it remains substantially less environmentally efficient per unit of Recall than DiFaR.

Note that this observation applies only to the inference phase. The substantial environmental costs associated with LLM training are not included as they fall outside the scope of this work. Furthermore, only HubLink (O) was executed on infrastructure, allowing for the direct measurement of energy consumption and carbon emissions, whereas all other approaches used the OpenAI API. Therefore, direct sustainability measurements for operations on external API servers are not feasible. Consequently, this difference limits the direct comparability of the sustainability figures between the locally run open-source model and the API-based models.

> **Answer to Q7:** *Our proposed HubLink approach generally exhibits a higher Recall performance at the cost of increased environmental impact compared to the most sustainable baseline approach, DiFaR. While HubLink (F) achieves a Recall nearly double that of DiFaR, it incurs a disproportionately larger environmental cost per unit of Recall. This highlights the need for careful consideration of the trade-off between maximizing Recall and minimizing environmental footprint.*

10.2. Evaluating Answer Alignment

In this section, we present the evaluation results that address the generation targets. We begin by analyzing the semantic and factual consistency of the generated answers, which relate to the generation target **GeT1**. Then we evaluate the relevance of the generated answers to the question and their alignment with the instructions provided in the question, which relates to **GeT2**. Finally, we assess the consistency of the generated answers with the retrieved context, which is relevant to **GeT3**.

*Note that unlike in the retrieval target evaluation, here only the HubLink variant (T) is evaluated. This is because the evaluation of the generated answers uses LLM-as-a-judge metrics, which incur additional costs for their computation. We have chosen to evaluate the HubLink variant that performed best in the retrieval target evaluation. Moreover, all the following evaluations are based on the graph variant **GV1**, also due to cost considerations. Finally, as detailed in Section 4.8.2.1, HubLink includes source citations and a corresponding reference list in the answer. Since these elements are absent from the reference answers, their presence likely penalizes similarity and precision metrics. Therefore, we do not include the reference list in the evaluation of the generated answers.*

Approach	FC-Reca.	FC-Prec.	FC-F1	Bert-Reca.	Bert-Prec.	Bert-F1
HubLink (T)	**0.543**	**0.301**	**0.361**	0.678	0.515	0.580
DiFaR	0.387	0.290	0.321	**0.702**	0.588	**0.635**
Mindmap	0.203	0.212	0.184	0.652	0.625	0.633
FiDeLiS	0.131	0.201	0.154	0.516	**0.629**	0.562
Approach	ROG-Reca.	ROG-Prec.	ROG-F1	Str. Sim.	Sem. Sim.	BLEU
HubLink (T)	**0.757**	0.298	0.373	0.261	0.761	0.105
DiFaR	0.674	0.374	**0.448**	**0.338**	**0.772**	**0.160**
Mindmap	0.487	0.432	0.397	0.296	0.682	0.105
FiDeLiS	0.195	**0.503**	0.251	0.202	0.499	0.046

Table 10.10.: Evaluation results for assessing the semantic and factual consistency of generated answers. The table includes various abbreviations: FC-Reca. (Factual Correctness Recall); FC-Prec. (Factual Correctness Precision); FC-F1 (Factual Correctness F1); Bert-Prec. (BERTScore Precision); Bert-Reca. (BERTScore Recall); Bert-F1 (BERTScore F1); ROG-Reca. (ROG-1 Recall); ROG-Prec. (ROG-1 Precision); ROG-F1 (ROG-1 F1); Str. Sim. (String Similarity); Sem. Sim. (Semantic Similarity). All metrics have been macro-averaged.

10.2.1. Semantical and Factual Consistency of Generated Answers

Building upon the observation from Table 10.1 that HubLink demonstrates superior performance in retrieving relevant triples compared to baseline KGQA approaches, this section evaluates the semantic and factual consistency of the answers generated based on these retrieved triples. In the following, we assess the evaluation results presented in Table 10.10. We begin by analyzing the Recall and similarity metrics, followed by an examination of the precision metrics.

10.2.1.1. Assessment of Recall and Similarity

The results in Table 10.10 illustrate a notable divergence from the retrieval performance assessment. Regarding Recall for factual correctness, HubLink achieves the highest value (0.543), although the advantage over competitors is less pronounced than observed in retrieval performance. Furthermore, compared to the retrieval Recall (0.754), the factual correctness Recall of HubLink is approximately 39% lower. This suggests limitations in preserving all retrieved facts during the answer generation process. However, this observation is contradicted by the particularly high ROUGE-1 Recall of 0.757, indicating that a substantial majority of the lexical items present in the reference answers are captured within the generated responses. Nevertheless, this does not necessarily imply that the generated answers include all the relevant facts from the retrieved triples. For example, if the generated answer includes many of the words also present in the reference answer, the ROGUE-1 Recall is high, even if the provided facts are wrong. Consequently, we conclude that HubLink does not fully retain all relevant information during answer generation.

In contrast, the baseline methods all achieved similar or higher factual correctness Recall values than their retrieval Recall. For instance, DiFaR achieved a factual correctness Recall value of 0.387, which closely aligns with its retrieval Recall (0.352), suggesting effective information transfer to the generation stage. Notably, Mindmap exhibits a factual correctness Recall of 0.203, largely exceeding its retrieval Recall (0.119). The same can be said for ROUGE-1 Recall, where all baseline methods achieved higher values than their retrieval Recall.

Regarding the BERTScore Recall, the results present a different pattern. DiFaR leads (0.702) with a slight edge over HubLink (0.678), followed by Mindmap (0.652) and FiDeLiS (0.516). A similar trend can be observed with BLEU scores, as well as the *String Similarity* and *Semantic Similarity* metrics, albeit with lower absolute values for BLEU. From the results, we can observe that DiFaR provides answers that are most similar to the expectation. Although HubLink does provide answers that are semantically related, the lower values in string similarity suggest that the generated answers tend to diverge, possibly because of the more comprehensive answers provided by the method.

Note that the lower absolute values for BLEU likely arise because the golden answers in the KGQA dataset were designed to be concise, only stating the facts asked for. Because BLEU measures exact n-gram overlap [76], this heavily penalizes any deviation in phrasing, structure, or additional information present. Since the scores are low, the LLMs seem to create more verbose answers.

10.2.1.2. Assessment of Precision

Analyzing precision for factual correctness, the score for HubLink (0.301) increased slightly over the retrieval precision (0.246). In stark contrast, baseline methods exhibit significantly higher factual correctness precision compared to their respective retrieval precision values. DiFaR achieves the highest factual correctness precision (0.290), surpassing HubLink despite having a very low retrieval precision (0.011). Mindmap (0.212 vs. 0.030) and FiDeLiS (0.201 vs. 0.052) show similar substantial increases from generation to retrieval precision.

However, for ROUGE-1 and BERTScore precision, HubLink records the lowest values among the evaluated methods (0.166 for ROG-Prec and 0.405 for Bert-Prec, respectively). This further highlights the structural and lexical differences between its generated answers and the reference targets.

10.2.1.3. Discussion on Results:

Several key observations arise from the evaluation of Table 10.10. First, the finding that factual correctness sometimes exceeds observations from the retrieval metrics warrants explanation, as generated answers should ideally be constrained by retrieved information. We attribute this to the implementation of the factual correctness metric within the RAGAS evaluation framework, which appears to assign partial credit based on the granularity of the answer. An answer deemed factually incomplete might still receive a positive evaluation if some parts are correct. For example, if the answer acknowledges the existence of a

Approach	Answer Relevancy	Instruction Following
HubLink (T)	**0.570**	**0.653**
DiFaR	0.203	0.312
Mindmap	0.545	0.388
FiDeLiS	0.432	0.388

Table 10.11.: Evaluation results assessing the alignment of generated answers with the intent and content of the question. All metrics have been macro-averaged.

publication without providing specific requested details, this contributes positively to the metric. This characteristic could contribute to higher scores of factual correctness for baseline methods relative to their retrieval performance.

Despite these limitations, valuable insights emerge. The collective results suggest that HubLink tends to generate more comprehensive, potentially overly elaborate answers compared to the reference targets. This may stem from its synthesis process, where the LLM integrates information from multiple retrieved sources, possibly leading to less concise outputs. Furthermore, the Recall scores are mediocre, suggesting that not all facts are transferred from the retrieved triples to the answer. Moreover, low precision and similarity scores indicate deviations in structure and the potential inclusion of extraneous details. Therefore, refining the integration of facts, the conciseness, and the focus of the generated answers of HubLink presents a direction for future improvement.

> **Answer to Q8:** *Compared to the baseline KGQA approaches, the answers generated by HubLink demonstrate limitations. The inclusion of facts from the retrieved triples into the generated answer is mediocre. Furthermore, lower scores in precision and similarity suggest that answers generated by HubLink may include additional, potentially unrequested, information and differ structurally from reference answers. This points to current limitations in the inclusion of facts, semantic consistency, and conciseness of answer generation.*

10.2.2. Generation of Relevant Answers

A crucial aspect of evaluating answer generation quality is determining whether the response is relevant to the posed question. Table 10.11 presents the results of the *answer relevancy* metric relevant to this aspect.

HubLink achieves the highest score of 0.570 in answer relevancy among the evaluated approaches. Mindmap follows closely with a score of 0.545, suggesting comparable effectiveness between these two methods in aligning generated responses with the intent of the question. FiDeLiS demonstrates moderate performance (0.432), whereas DiFaR shows considerably lower relevancy (0.203).

These findings indicate that although HubLink leads in answer relevancy relative to the baselines, its absolute score suggests that a notable portion (43%) of its generated answers

may not be optimally aligned with the question, highlighting the scope for improvement. However, it is critical to underscore that the Answer Relevancy metric assesses the perceived alignment between the question and the topic or intent of the answer, independent of the factual accuracy. Consequently, a response could be deemed relevant yet contain factual inaccuracies or hallucinations. The factual correctness aspect is specifically addressed in Table 10.10.

> **Answer to Q9:** *HubLink demonstrates the strongest performance among the evaluated methods in generating answers that align with the semantic intent of scholarly questions, as measured by answer relevancy. However, its absolute performance indicates limitations, suggesting that further refinement is necessary to consistently ensure optimal semantic alignment between questions and generated answers.*

10.2.3. Following the Instructions provided in the Question

Beyond requesting specific information, questions may include explicit instructions regarding the desired answer format or structure. The KGQA dataset that has been used incorporates such questions derived from complex retrieval operations. This requires the retriever, for instance, to present results in a specific order or perform aggregations. The ability of each approach to comply with these requirements is evaluated using the *Instruction Following* metric, with results presented in Table 10.11.

The results in Table 10.11 indicate that HubLink substantially outperforms the baseline KGQA approaches in adhering to question instructions by achieving a score of 0.653, which is approximately 68% higher than the scores of the next-best-performing methods, Mindmap and FiDeLiS (both 0.388). DiFaR demonstrated lower performance on this metric (0.312).

Despite its relative advantage, the absolute performance of HubLink reveals limitations, as the score of 0.653 implies that the system did not fully adhere to instructions in approximately one third (34%) of the cases. This indicates that while HubLink demonstrates a significantly stronger capability for instruction following compared to the baselines, further refinement of its generation process is warranted to improve reliability in this aspect.

> **Answer to Q10:** *HubLink exhibits a significantly superior ability to follow specific instructions embedded within scholarly questions compared to the baseline KGQA approaches evaluated. Nonetheless, its absolute performance indicates that adherence to instructions is not fully consistent, highlighting the need for further enhancements in the answer generation mechanism to ensure instructions are followed more reliably.*

10.2.4. Consistency of the Generated Answers to the Retrieved Context

A critical requirement for trustworthy answer generation, particularly when using LLMs, is to ensure that the output is strictly grounded in the retrieved context. The generated answer must refrain from introducing extraneous information, and all presented assertions

Approach	Faithfulness
HubLink (T)	0.445
DiFaR	**0.645**
Mindmap	0.396
FiDeLiS	0.112

Table 10.12.: Evaluation results assessing how consistent the generated answer is with the retrieved contexts.

should be directly verifiable against the source data. Table 10.12 presents the evaluation results using the *Faithfulness* metric, designed to measure conformity with the retrieved context.

The data reveals that DiFaR achieves the highest faithfulness score (0.645), indicating strong adherence to its retrieved context. The score of HubLink (0.445) is notably lower, comparable to the performance of Mindmap (0.396), while FiDeLiS exhibits substantially lower faithfulness (0.112).

These results suggest that HubLink exhibits notable limitations in constraining its answers solely to the provided context. The lower faithfulness score of HubLink compared to DiFaR indicates that the latter is more effective in ensuring that generated answers are strictly grounded in the retrieved context. This confirms findings from previous generation metrics, as it underscores that the current answer generation strategy in HubLink constitutes a limitation and necessitates refinement to enhance strict factual grounding alongside overall answer quality.

> **Answer to Q11:** *The generated answers of HubLink demonstrate weaker grounding in the retrieved context compared to the baseline with the highest performance, DiFaR. Therefore, improving the faithfulness of responses to the retrieved context and minimizing potentially unsupported claims is an area that requires improvement.*

10.3. Discussion on Evaluation Results

In the following, we discuss the interpretation of the evaluation results concerning our proposed HubLink approach. This discussion focuses on the two primary aspects evaluated: data retrieval from the KG and the subsequent answer generation. Furthermore, as the baseline approaches yielded notably low scores, we will discuss their results separately.

10.3.1. Analysis of Retrieval Performance

Our evaluation indicates that HubLink presents a considerable advancement in retrieving scholarly information from RKGs compared to existing baseline methods, positioning it as a

promising approach for scholarly literature searches. The consistently high recall observed across our tests underscores that the core strength of HubLink lies in its capacity to identify and retrieve a comprehensive set of relevant triples from the KG. This capacity remains robust when HubLink is applied to diverse graph structures, showcasing the schema-agnostic characteristic of the approach. The results further demonstrate the capability of HubLink to retrieve relevant information even when questions necessitate traversing multiple hops or when information is spread across broader paths. This multi-hop retrieval capability is particularly important for scholarly literature search tasks, where questions necessitate multi-hop reasoning to connect diverse pieces of information.

The evaluation also demonstrates that HubLink can effectively handle a range of retrieval operations, including basic factual lookups and moderately complex operations such as aggregation and ranking. In addition, the results show that HubLink can operate economically during the retrieval process. Specifically, a faster configuration of the approach achieves performance comparable to that of a more complex configuration while offering the benefits of reduced runtime, lower LLM token costs, and diminished environmental impact. This efficiency is particularly relevant for real-world applications, where efficiency and cost-effectiveness are crucial considerations.

However, the evaluation also highlights areas where HubLink faces challenges. Although HubLink achieves high recall, the moderate precision and ranking scores indicate difficulty in distinguishing relevant triples from less pertinent ones within the retrieved set. This suggests that HubLink retrieves a broader set of information than required, which includes noise alongside the relevant context. This challenge becomes even more pronounced for questions that require complex logical operations, such as negation or superlatives, and for use cases involving less structured content data. These limitations suggest that, while HubLink excels at retrieving information for answering scholarly questions, downstream filtering or reranking mechanisms may be necessary to refine the results for optimal precision and ranking quality.

10.3.2. Analysis of Answer Generation Performance

With regard to the answer generation performance, HubLink demonstrates improved capacity over baseline approaches to generate relevant answers and adhere to specific instructions embedded within the query. This suggests that HubLink captures user intent more effectively, even for complex tasks.

However, our analysis shows that the translation of retrieved facts into consistently high-quality answers presents limitations. Our evaluations indicate that HubLink only partially incorporates the retrieved factual knowledge into its generated answers. Furthermore, the absolute recall scores for answer generation are lower than those observed during retrieval, suggesting that not all relevant information from the retrieved context is integrated into the final answers. Moreover, lower scores in precision and similarity comparisons suggest that the answers generated by HubLink tend to be less concise and may deviate structurally from the expected reference answers. A reduced faithfulness score also indicates this trend.

Overall, the generation component exhibits a tendency towards producing comprehensive yet potentially overly verbose answers rather than consistently concise presentations of information. Therefore, an important direction for future improvement is the refinement of the generation strategy to produce more focused and concise answers while still maintaining accuracy and relevance to the context.

10.3.3. Analysis of Baseline Performances

Our experimental results reveal that the evaluated baseline methods generally exhibit substantially lower retrieval performance compared to our proposed HubLink approach. In particular, FiDeLiS and Mindmap demonstrated low performance in the retrieval experiments, while StructGPT and ToG struggled to answer any question during the parameter selection process (see Chapter 9). Consequently, we did not use StructGPT and ToG in further testing. In the following subsections, we examine the specific characteristics and limitations of each baseline method to understand these performance differences.

10.3.3.1. DiFaR

Among the baseline approaches evaluated, DiFaR [99] achieved the best overall retrieval performance across most metrics. This approach, similar to HubLink, is based on leveraging embeddings for retrieval tasks. The relative success of DiFaR compared to the non-embedding-based baselines suggests potential advantages of embedding strategies for the types of questions and graph structures used in our experiments.

However, HubLink consistently outperforms DiFaR. This comparison highlights that, while a general embedding-based strategy is beneficial, the specific retrieval mechanisms and greater complexity introduced by HubLink yield a justifiable improvement in performance over the DiFaR approach.

10.3.3.2. Mindmap

The Mindmap approach [96] operates by constructing evidence subgraphs. However, it encountered difficulties in our experiments, demonstrating a limited ability to answer the questions correctly. During the analysis of the algorithm, we observed a substantial limitation that renders the Mindmap approach unsuitable for scholarly literature search in our experimental setup. The following example illustrates the issue with a representative question from our dataset. The fact that Mindmap is unable to answer this question highlights its fundamental limitation, which hinders its performance on many other questions in the dataset.

Given is the following question: “Who are the authors of the paper ’A Taxonomy of Blockchain-Based Systems for Architecture Design’?”. When processing this question, Mindmap first extracts entities from the query. A plausible extraction would yield the terms

Authors, *Paper*, and the title *A Taxonomy of Blockchain-Based Systems for Architecture Design*. After this term recognition process, the approach queries the graph to collect entities that match these terms with the highest similarity scores. These entities are then used to build evidence subgraphs.

Mindmap constructs two types of evidence subgraphs. The first is the path-based subgraph, which finds the shortest paths between the identified entity nodes and stores them. The second is the neighbor-based subgraph, which collects all one-hop neighbors for each identified entity. However, in our example, only one term in the question corresponds directly to an entity in the graph with a meaningful match: the title of the publication. The other two extracted terms (Authors, Paper) represent semantic types or concepts rather than specific entity nodes within the graph structure, so no corresponding entities that are meaningful are found. Consequently, the only way for Mindmap to answer the question correctly would be if the authors were stored as immediate neighbors of the entity containing the title of the publication. However, in the ORKG this is not the case, as the triples of the authors are stored deeper in the graph and are therefore not collected by the retrieval.

Consequently, Mindmap performs poorly in our experiments because many questions in the KGQA dataset provide only a single known entity and ask for another unknown entity. To be able to correctly answer these questions, it is required to find the path from the known entity to the unknown entity, which the Mindmap approach is unable to do since it builds paths only between entities provided in the question. This highlights a fundamental limitation of the Mindmap retriever and explains why many questions in the KGQA dataset have not been answered correctly.

10.3.3.3. Beam Search Retrievers

The StructGPT [92], ToG [88], and FiDeLiS [93] approaches all rely on the beam search algorithm for graph exploration. Our experiments suggest that these approaches perform poorly on the KGQA dataset. During the analysis of these algorithms, we discovered two major issues that explain the low performance, which are discussed below.

Local Information Deficit: The beam search algorithm explores the knowledge graph by iteratively expanding a limited set of the most promising nodes or paths up to a predefined depth. A key challenge arises from its decision-making process: the choice of which paths to retain in the beam at each step is based on local information associated with the current nodes or their immediate neighbors. If the relevance of a path towards the final answer is not apparent from this local context, the path may be pruned, even if further exploration along that path would eventually lead to the correct answer.

To illustrate, consider the question “Which papers have used an interview as a method for their evaluation?”. A potentially relevant path might be structured as:

$$\textit{Paper Title Node} \xrightarrow{\text{hasContribution}} \textit{Contribution Node} \xrightarrow{\text{usesMethod}} \textit{'Interview' Node}$$

When the beam search exploration reaches a specific *Paper Title Node*, it must decide whether to keep exploring paths originating from it based primarily on information available at that node. The title itself, or even the immediate 1-hop neighbors, may provide insufficient evidence that this specific path will lead to the target *'Interview' node* when traversing further. As suggested by our analysis of the number of hops in Section 10.1.5, this issue is even more pronounced for questions that require more than three hops to arrive at the answer. Because most of the questions in our experiment require a larger number of hops, this exaggerates the issue, leading to the premature pruning of relevant paths and degrading the overall performance of these retrievers.

Stochastic Selection: The second challenge comes from the sheer breadth of the graph. Because only a limited number of entities can be expanded at each iteration, the selection of the most relevant candidates is critical. StructGPT and ToG rely on an LLM to classify the relevance of predicates and then consider entities connected via the selected predicates for the next beam exploration. If this set of candidate entities exceeds the predefined beam width threshold, the approaches randomly prune the entities to reduce the size of the candidate set. This introduces non-determinism and the risk of discarding correct entities or paths purely by chance.

FiDeLiS addresses this challenge by using embedding similarity to score and select entities instead of random sampling. This results in more deterministic and often more relevant entity selection, contributing to its generally better performance compared to StructGPT and ToG in our results. However, FiDeLiS remains constrained by the first limitation, which hinders its effectiveness on questions that require deeper graph traversal where relevance is not immediately apparent.

10.4. Threats to Validity

In this section, we discuss the threats to validity for our experiment results. For this discussion, we are using the descriptions and checklists provided by Wohlin et al. [153, pp. 131–140] who propose to discuss the concepts *Conclusion Validity*, *Internal Validity*, *Construct Validity*, and *External Validity*. However, their checklist applies to experiments with human subjects, which is not the case for our experiments. Consequently, we only include the points that are relevant for our experimental setup. In addition, we also include the concepts *Credibility*, *Dependability*, and *Confirmability* proposed by Feldt and Magazinius [154].

10.4.1. Conclusion Validity

The threats to conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about the relations between the treatment and the outcome of an experiment.

Reliability of Measures describes that the outcome of a measurement should be the same for equal inputs. In our experiments, we use both traditional metrics and LLM-as-a-Judge metrics. For the LLM-based metrics, there is no guarantee that they will always produce the same evaluation for identical inputs. To mitigate this issue, we employ RAGAS [74], a specialized evaluation framework for LLM-as-a-Judge metrics, because one of the main objectives of the framework is to enhance the reliability and reproducibility of these metrics.

Reliability of Treatment Implementation considers whether the treatments are applied correctly. We developed the SQA framework, which allows us to maintain consistent configurations while selectively varying the treatments for the experiments. Consequently, we do not see any issues with the implementation of the treatments.

Random Irrelevancies in the Experimental Setting are concerned with random elements outside the experimental setting that disturb the results. For our experiments, we use a server provided by the institute that is shared among several users. This shared usage may introduce random disturbances, such as variations in execution time when others place a high load on the server during our experiments. To mitigate this, we verify that the server is not under load before starting the experiments.

10.4.2. Internal Validity

Threats to internal validity are those influences that can affect dependent variables with respect to causality without the knowledge of the researcher. As such, they threaten the conclusion about a possible causal relationship between the treatment and the outcome. From the checklist provided by Wohlin et al. [153, pp. 133–134], we only see instrumentation as relevant for our experiments.

Instrumentation is about considering the quality of the artifacts used for the execution of the experiment that may negatively affect the results. To realize the execution of the experiments, we have implemented the SQA framework and adapted baseline retrievers based on their descriptions provided by the authors and the available code to work with our framework. As such, there is a risk that if the implementations are poorly designed and executed, the results of the experiments are negatively affected. To mitigate this risk, the SQA framework has undergone an architectural review and two rounds of code reviews with domain experts. Furthermore, the implementations of the baseline retrievers have been done with minimal changes to the original code (see Section 7.4).

10.4.3. Construct Validity

Construct validity ensures that the metrics and methods that we have used accurately capture the intended evaluation constructs that we outline in Section 8.4. The following points from the checklist provided by Wohlin et al. [153, pp. 146–137] are relevant to our experiments.

Inadequate Pre-operational Explication of Constructs relates to the issue that the constructs are not sufficiently described before they are translated into measures or treatments. We do not see a risk of inadequate pre-operational explication of the constructs because the constructs that we are evaluating are based on the evaluation framework RGAR [69] and multiple surveys about RAG evaluation (see Section 2.6).

Mono-Operation Bias is concerned with the underrepresentation of constructs due to a singular independent variable, case, subject, or treatment. Although this singularity is not the case for our experiments, we still see a considerable threat of the underrepresentation of constructs. This is because we have only included a subset of the possible configurations for each retriever. However, this was necessary to keep the experiments within a reasonable scope.

Mono-Method Bias is concerned with the risks of using only one type of measure or observation, which can become an issue if measurement bias occurs. However, as we are using established metrics (e.g., recall, precision) in the field and have tested them prior to the experimentation, we do not see the need to conduct multiple measurements for the same constructs. There is a small risk that the measurement of the faithfulness and relevance of the generated answers is underrepresented. However, we believe that the metrics chosen from the RAGAS framework are representative of the constructs. Furthermore, regarding singular observations, we are only performing each experiment once. Hence we only have a singular observation for each treatment. However, because we expect the results to be mostly consistent across multiple runs, we do not see this as a risk to our experiments.

Interaction of Different Treatments describes the risk of having one subject participate in more than one study, which could lead to a treatment interaction. In our experiments, we are not using human subjects, hence this is not a risk for our experiments. However, there is a risk that the results of the experiments are affected by the interaction of different treatments. This is because we are using the OFAT method to evaluate the effect of each factor on the outcome. However, this was necessary to keep the experiments within a reasonable scope. To reduce this risk, we have carefully considered the parameters of the retrievers that interact with each other.

Restricted Generalizability Across Constructs is about treatments that positively affect one construct but unintentionally negatively affect another construct. We do not see this as a risk for our experiments because the constructs are all evaluated at the same time, which makes it possible to see the trade-offs of treatments on each of the constructs.

10.4.4. External Validity

The external validity ensures that the results of the experiment can be generalized beyond the experimental setting.

Interaction of Selection and Treatment is concerned with the effect that the subject population that is used does not represent the population of interest. In our experiments, there is a risk that the questions in our KGQA dataset do not represent actual questions of interest that a

researcher would ask. To mitigate this risk, we have generated the questions based on a question taxonomy of desired question types and six use cases for the literature research task.

Interaction of Setting and Treatment describes the issue of not using an experimental setting or tools that are representative of the real world. We mitigated this risk by developing the SQA framework according to the state-of-the-art approach applied inQA systems, which is the RAG approach. Furthermore, we researched common evaluation metrics for RAG systems and applied them to our experiments using the formulas and implementations provided.

10.4.5. Credibility

The credibility describes whether there is a risk that the results of the experiments are not true. We do not see a risk of credibility in our experiments. The experiments are carried out in a unified framework, the SQA framework, which ensures that all treatments are applied under consistent conditions. This is further achieved by maintaining identical experimental settings and also by using the same hardware and software environment for all experiments. This makes it highly likely that any variance in outcomes is attributable to treatments rather than uncontrolled external factors.

10.4.6. Dependability

Dependability concerns the risk that the results of the experiments are not repeatable. There is a risk that the results of the experiments cannot be reproduced exactly, as we are working with LLMs, which are inherently non-deterministic. However, we expect the performances of the KGQA approaches to be similar across multiple runs and the overall trends to be consistent. To allow for high reproducibility, the SQA framework is able to exactly reproduce the same experimental setting, reducing the risk to only the randomness of the LLMs.

10.4.7. Confirmability

The confirmability is concerned with the risk that the results of the experiments are not based on the data but on the bias of the researcher. Based on the evaluation of the experimental results, we acknowledge the risk of bias in the interpretation of the results. This is because the interpretation of the results has been made by the understanding of the author of each of the constructs and their metrics. However, we mitigate this risk by clearly presenting the results in multiple diagrams, tables, and in a replication package. In addition, we thoroughly discuss the results. This allows the reader to draw their own conclusions.

Furthermore, the baselines have been chosen by the author of this paper and there is a risk that the selected retrievers are not representative of the state of the art in QA systems

for the literature research task. To mitigate this risk, we have carefully reviewed the most recent surveys [155, 23, 22, 13, 21, 14] on KGQA and selected retrievers that were applicable for our task. More details on the selection of the retrievers can be found in Section 7.4.

11. Conclusion

This master thesis addressed a research gap in applying training-free and schema-agnostic KGQA approaches to the task of scholarly literature search. Our research aimed to overcome the limitations associated with existing approaches, specifically their reliance on fixed schemas and training data (**P1**), and the lack of a standardized taxonomy for assessing capabilities of KGQA systems (**P2**).

Consequently, the master thesis introduced HubLink (**C1**), a novel approach that represents a significant step toward schema-agnostic and training-free KGQA retrieval. By conceptually decomposing the graph into structures termed *hubs*, which aggregate the knowledge from scholarly publications, HubLink enables a modular and source-aware retrieval process. Specifically, the ability of the approach to transparently trace the origin of information during inference provides a key requirement for scholarly literature retrieval. Furthermore, the embedding-based methodology employed by HubLink leverages the semantic connections captured by LLMs without necessitating explicit training data or adhering to fixed graph schemas. This independence makes the approach inherently adaptable to dynamic and evolving graph structures.

Our evaluation results demonstrate that HubLink substantially improves the relevance and accuracy of retrieved contexts compared to state-of-the-art baseline KGQA approaches, many of which struggled in the scholarly domain application. The evaluation further highlights that HubLink can be successfully applied to six specific use cases for the scholarly literature task, showing the potential of the approach to improve the efficiency of scientific communication. Although the evaluation demonstrated the clear advantages of HubLink in retrieving relevant information, it also identified specific areas for future enhancement. In particular, the results suggest that there are opportunities to refine the precision of the retrieved context and their ranking by relevance. Furthermore, the process of generating coherent and precise responses based on the retrieved context warrants further investigation and development. Future work should therefore focus on improving these aspects to fully realize the potential of the approach.

Complementing the HubLink approach, this thesis proposed a new taxonomy for KGQA specifically tailored to the literature search task (**C2**). Developed through a systematic and operationalized construction process (**C2.2**), this taxonomy provides a structured framework to classify questions based on characteristics relevant to retrieval performance and scholarly information needs. The validation process, which includes the application to research questions and comparison with existing dataset-specific classifications, confirms that the taxonomy addresses a notable gap by offering a more structured and fine-grained framework for understanding and assessing KGQA system capabilities in the scholarly domain. The

usefulness of this taxonomy is further demonstrated through its successful application in the construction of new KGQA datasets (**C3**) for the ORKG. These datasets were used to conduct the comprehensive evaluation of HubLink and the baseline methods across diverse question types.

To conclude, current scholarly practices embed scholarly findings within unstructured documents, rendering data extraction and targeted search cumbersome. In contrast, RKGs store scholarly findings in a structured network. The application of HubLink to retrieve relevant data from RKGs in a KGQA setting enables direct and precise access to scholarly information, reducing manual effort. This paradigm significantly reduces the effort required by researchers to locate relevant scholarly findings. Consequently, this work makes a significant contribution to the advancement of research in scholarly KGQA, offering a promising path towards more efficient and effective scientific discovery and communication in the future.

11.1. Research Questions Revisited

The primary research goal was to design a new schema-agnostic and training-free retrieval approach capable of effectively retrieving scholarly knowledge from RKGs, along with developing a systematic taxonomy to assess the capabilities of such systems. In this section, we revisit the two primary research questions that we defined and successfully addressed based on our contributions.

> ***RQ1*** *How can a schema-agnostic retrieval algorithm leveraging an RKG and a pre-trained LLM be developed for the KGQA setting to effectively integrate diverse scholarly sources, adapt to evolving schemas and account for the provenance of information during retrieval without relying on training data?*

We found that the most effective way is to implement an embedding-based approach, which we refer to as HubLink. This approach does not require any further training data as it relies on the inherent semantic connections between words and sentences. This makes it particularly suitable for handling evolving schemas and unseen entities, as no knowledge about the schema is required during retrieval. Moreover, by dividing the graph into hub structures, with each hub representing a publication, the source of the information is taken into account during retrieval. By doing so, an overreliance on a single source is mitigated and the LLM is encouraged to integrate diverse perspectives.

> ***RQ2*** *How can existing general QA and KGQA taxonomies be synthesized and extended to form a comprehensive taxonomy tailored to define the characteristics of questions posed to KGQA retrieval systems for the literature search task?*

We found that the creation of such a taxonomy can be successfully achieved through the taxonomy construction methodology introduced in this thesis. This systematic approach involves extracting relevant concepts from the existing literature, followed by clustering

and relevance assessment phases to initialize a synthesized taxonomy, which is then incrementally refined and validated. The utility of this methodology was confirmed through the successful development of a specific taxonomy designed to classify question characteristics within the domain of KGQA systems applied to literature searches. The practical value of the resulting taxonomy was further underscored during the creation of KGQA benchmarking datasets. Here, the taxonomy served as a framework to ensure that the datasets cover a diverse range of question types, enabling a comprehensive assessment of different system capabilities during evaluation.

11.2. Future Work

This thesis presented HubLink, a novel schema-agnostic and training-free KGQA retrieval approach, alongside a new taxonomy for classifying KGQA questions within the scholarly domain. Building upon these contributions and the findings presented, this section outlines potential directions for future work. These directions are organized based on their relevance to the HubLink retrieval approach as well as the developed taxonomy and its construction process.

11.2.1. Schema-Agnostic and Training-Free KGQA Retrieval

Future work related to the HubLink retrieval approach could focus on the following areas:

Improving Answer Generation Our evaluation results presented in Section 10.2 indicate limitations in the alignment between retrieved contexts and the final generated answers for HubLink. Future work could investigate methods to enhance this alignment. Potential strategies include advanced prompt engineering techniques aimed specifically at improving the accuracy of the answer generation process relative to the provided context or exploring alternative synthesis methods beyond single-prompt generation.

Further Evaluations As described in Section 8.3.2, the KGQA datasets that were used in this thesis reside at the label-based granularity level. Future work could apply HubLink to other datasets that involve the retrieval of knowledge from RKGs at a different level of granularity to verify the results provided in this thesis. This includes the evaluation of the *linking* feature provided by HubLink, which was not useful in our evaluation scenario, due to the label-based abstraction level in our data. However, if data are used at other granularity levels, the linking feature could be beneficial.

Improving Relevancy Ranking A limitation that our evaluation of retrieval performance in Section 10.1 reveals is the observed limitation of HubLink to rank relevant context higher than irrelevant context. Consequently, future work could focus on improving this limitation, possibly through refined prompt strategies for relevance assessment or the integration of dedicated reranking steps following the retrieval phase.

Extension of Hub Content One interesting research direction for enhancing HubLink involves the extension of the hubs with further information. Future work could explore expanding the index with information such as summaries or precomputations of numerical aggregations derived from associated papers. Such extensions possess the potential to enrich the contextual information available during retrieval and synthesis, possibly leading to more comprehensive and accurate answers.

Numerical Constraints The processing of numerical constraints, such as dates, ranges of values, or specific metric comparisons, presents a potential challenge for embedding-based retrieval approaches like HubLink, as highlighted in previous research [156]. Although we do have temporal constraints in our evaluation and the retriever demonstrates that it is able to consider them during inference, these do not involve complex time spans. Future work should explicitly evaluate the performance of HubLink on questions requiring complex numerical reasoning or filtering.

Document-based Setting In our work, we primarily focused on the retrieval of knowledge stored in the graph. Future work could evaluate HubLink in a document retrieval setting. In such a setting, HubLink uses the KG as a supporting structure similar to other prominent approaches in the literature [157, 158]. The use of the linking capability of HubLink could be utilized in such a scenario to retrieve relevant document text chunks and return them. In this case, the knowledge information from the graph serves as a preliminary step to allow source-aware retrieval. We have already conducted preliminary work towards this evaluation, including the implementation of LightRag [158] and Microsoft GraphRAG [157] as baselines, the preparation of the evaluation code of the SQA system to work with document-based approaches, and the implementation of an algorithm for extracting structured knowledge from scientific papers. However, the evaluation could not be completed in the time frame of this thesis.

Other Graphs The evaluation in this thesis focused on the ORKG. Future studies could investigate the applicability and performance of HubLink on other RKGs to assess its robustness and utility across different knowledge representations within the scholarly domain. Furthermore, as argued in Section 4.10, the design principles of HubLink suggest potential applicability beyond the scholarly domain, particularly for large and heterogeneous KGs. Investigating the scalability and effectiveness of HubLink on various KGs in other domains represents another important research direction to determine the generalization of the approach.

11.2.2. Taxonomy Construction and Application in KGQA

Regarding the proposed taxonomy and its construction process, future work could explore the following directions:

Taxonomy Application in Diverse Domains The taxonomy developed in this work proved useful in constructing a KGQA dataset tailored to scholarly literature searches. However, its potential applicability extends beyond this specific context. Future work could explore the utility of this taxonomy for characterizing questions or guiding dataset creation in different KGQA domains. This may necessitate adaptations to accommodate domain-specific question types or information needs but would test the broader relevance of the classification scheme.

Continuation of Literature Survey The construction of the KGQA retrieval taxonomy involves a comprehensive literature survey that was stopped prematurely after the second iteration. Although we argue that most interesting approaches that stem from the seed papers are considered, future work could extend the search by incorporating more sources from different domains. This process can be easily extended using the artifacts provided in our replication package [124].

Construction of new Taxonomies This thesis introduced a systematic process for constructing taxonomies. To validate its broader utility and identify potential limitations, further application by researchers in different contexts would be valuable. Future work could involve employing this construction methodology to develop taxonomies in other subject areas or for different purposes. Such effort would help assess the robustness, adaptability, and generalizability of the proposed taxonomy construction process itself.

Bibliography

[1] Mohamad Yaser Jaradeh et al. "Open Research Knowledge Graph: Next Generation Infrastructure for Semantic Scholarly Knowledge". In: *Proceedings of the 10th International Conference on Knowledge Capture*. ACM, Sept. 23, 2019, pp. 243–246. DOI: 10.1145/3360901.3364435.

[2] Sören Auer et al. "Towards a Knowledge Graph for Science". In: *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*. June 25, 2018, pp. 1–6. ISBN: 978-1-4503-5489-9. DOI: 10.1145/3227609.3227689.

[3] H. Sompel and C. Lagoze. "All aboard: toward a machine-friendly scholarly communication system". In: The Fourth Paradigm. 2009.

[4] Jeroen Bosman et al. "The Scholarly Commons - principles and practices to guide research communication". In: Sept. 15, 2017. DOI: 10.31219/osf.io/6c2xt.

[5] Lutz Bornmann, Robin Haunschild, and Rüdiger Mutz. "Growth rates of modern science: a latent piecewise growth curve approach to model publication numbers from established and new literature databases". In: *Humanities and Social Sciences Communications* 8.1 (Oct. 7, 2021), pp. 1–15. ISSN: 2662-9992. DOI: 10.1057/s41599-021-00903-w.

[6] Dan Li et al. "Unsupervised Dense Retrieval for Scientific Articles". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*. EMNLP 2022. Association for Computational Linguistics, Dec. 2022, pp. 313–321. DOI: 10.18653/v1/2022.emnlp-industry.32.

[7] Linyao Yang et al. *Give Us the Facts: Enhancing Large Language Models with Knowledge Graphs for Fact-aware Language Modeling*. Jan. 30, 2024. URL: http://arxiv.org/abs/2306.11489 (visited on 07/09/2024).

[8] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Apr. 12, 2021. URL: http://arxiv.org/abs/2005.11401 (visited on 07/09/2024).

[9] Xuantao Lu and Xingwu Hu. "Dense Retrieval for Efficient Paper Retrieval in Academic Question Answering". In: KDD 2024 OAG-Challenge Cup. July 21, 2024. URL: https://openreview.net/forum?id=iH203OQIU5 (visited on 05/27/2025).

[10] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. Mar. 27, 2024. URL: http://arxiv.org/abs/2312.10997 (visited on 07/09/2024).

[11] Jiawei Chen et al. *Benchmarking Large Language Models in Retrieval-Augmented Generation*. Dec. 20, 2023. DOI: 10.48550/arXiv.2309.01431. arXiv: 2309.01431[cs].

[12] Shilpa Verma et al. "Scholarly knowledge graphs through structuring scholarly communication: a review". In: *Complex & Intelligent Systems* 9.1 (Feb. 1, 2023), pp. 1059–1095. ISSN: 2198-6053. DOI: 10.1007/s40747-022-00806-6.

[13] Boci Peng et al. *Graph Retrieval-Augmented Generation: A Survey*. Sept. 10, 2024. DOI: 10.48550/arXiv.2408.08921. arXiv: 2408.08921[cs].

[14] Shirui Pan et al. "Unifying Large Language Models and Knowledge Graphs: A Roadmap". In: *IEEE Transactions on Knowledge and Data Engineering* 36.7 (July 2024), pp. 3580–3599. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2024.3352100.

[15] Debayan Banerjee. "Knowledge Graph Question Answering With Generative Language Models". ISBN: 9798346770350. PhD thesis. Germany: Universitaet Hamburg (Germany), 2024. 132 pp. URL: https://www.proquest.com/docview/3143985996/abstract/3893F0C5845D4D1DPQ/1 (visited on 01/21/2025).

[16] Nilesh Chakraborty et al. "Introduction to neural network-based question answering over knowledge graphs". In: *WIREs Data Mining and Knowledge Discovery* 11.3 (2021), e1389. ISSN: 1942-4795. DOI: 10.1002/widm.1389.

[17] Nilesh Chakraborty et al. *Introduction to Neural Network based Approaches for Question Answering over Knowledge Graphs*. July 22, 2019. DOI: 10.48550/arXiv.1907.09361.

[18] Mohammad Yani, Adila Alfa Krisnadhi, and Indra Budi. "A better entity detection of question for knowledge graph question answering through extracting position-based patterns". In: *Journal of Big Data* 9.1 (June 17, 2022), p. 80. ISSN: 2196-1115. DOI: 10.1186/s40537-022-00631-1.

[19] Bowen Jin et al. *Large Language Models on Graphs: A Comprehensive Survey*. Feb. 1, 2024. URL: http://arxiv.org/abs/2312.02783 (visited on 07/09/2024).

[20] Zhangyin Feng et al. *Trends in Integration of Knowledge and Large Language Models: A Survey and Taxonomy of Methods, Benchmarks, and Applications*. Dec. 7, 2023. URL: http://arxiv.org/abs/2311.05876 (visited on 07/09/2024).

[21] Yuhan Li et al. *A Survey of Graph Meets Large Language Model: Progress and Future Directions*. Apr. 24, 2024. DOI: 10.48550/arXiv.2311.12399. arXiv: 2311.12399[cs].

[22] Garima Agrawal et al. *Can Knowledge Graphs Reduce Hallucinations in LLMs? : A Survey*. Mar. 15, 2024. URL: http://arxiv.org/abs/2311.07914 (visited on 07/09/2024).

[23] Tyler Thomas Procko and Omar Ochoa. "Graph Retrieval-Augmented Generation for Large Language Models: A Survey". In: *2024 Conference on AI, Science, Engineering, and Technology (AIxSET)*. 2024 Conference on AI, Science, Engineering, and Technology (AIxSET). Sept. 2024, pp. 166–169. DOI: 10.1109/AIxSET62544.2024.00030. (Visited on 01/07/2025).

[24] Debayan Banerjee et al. *DBLP-QuAD: A Question Answering Dataset over the DBLP Scholarly Knowledge Graph*. Mar. 29, 2023. URL: http://arxiv.org/abs/2303.13351 (visited on 07/09/2024).

[25] Tilahun Abedissa Taffa and Ricardo Usbeck. “Leveraging LLMs in Scholarly Knowledge Graph Question Answering”. In: (2023). DOI: 10.48550/ARXIV.2311.09841.

[26] Jens Lehmann et al. “Large Language Models for Scientific Question Answering: An Extensive Analysis of the SciQA Benchmark”. In: *The Semantic Web*. Ed. by Albert Meroño Peñuela et al. Cham: Springer Nature Switzerland, 2024, pp. 199–217. ISBN: 978-3-031-60626-7. DOI: 10.1007/978-3-031-60626-7_11.

[27] Longquan Jiang, Xi Yan, and Ricardo Usbeck. “A Structure and Content Prompt-based Method for Knowledge Graph Question Answering over Scholarly Data”. In: QALD/SemREC@ISWC. Jan. 1, 2023. URL: https://openreview.net/forum?id=Mxc32gezHC (visited on 05/27/2025).

[28] Mohamad Yaser Jaradeh, Markus Stocker, and Sören Auer. *Question Answering on Scholarly Knowledge Graphs*. June 2, 2020. DOI: 10.48550/arXiv.2006.01527. arXiv: 2006.01527.

[29] Yu Gu et al. *Knowledge Base Question Answering: A Semantic Parsing Perspective*. Oct. 24, 2022. DOI: 10.48550/arXiv.2209.04994. arXiv: 2209.04994[cs].

[30] Tanik Saikh et al. “ScienceQA: a novel resource for question answering on scholarly articles”. In: *International Journal on Digital Libraries* 23.3 (Sept. 1, 2022), pp. 289–301. ISSN: 1432-1300. DOI: 10.1007/s00799-022-00329-y.

[31] Mohnish Dubey et al. “LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia”. In: *The Semantic Web – ISWC 2019*. Ed. by Chiara Ghidini et al. Cham: Springer International Publishing, 2019, pp. 69–78. ISBN: 978-3-030-30796-7. DOI: 10.1007/978-3-030-30796-7_5.

[32] Xin Li and Dan Roth. “Learning Question Classifiers”. In: *COLING 2002: The 19th International Conference on Computational Linguistics*. 2002. DOI: 10.3115/1072228.107237.

[33] Amit Singhal et al. “AT&T at TREC-8”. In: Nov. 1, 1999.

[34] Lisa Ehrlinger and Wolfram Wöß. “Towards a Definition of Knowledge Graphs”. In: International Conference on Semantic Systems. 2016.

[35] David Wood, Markus Lanthaler, and Richard Cyganiak. *RDF 1.1 Concepts and Abstract Syntax*. Publisher: W3C. Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (visited on 03/15/2015).

[36] Shaoxiong Ji et al. “A Survey on Knowledge Graphs: Representation, Acquisition, and Applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.2 (Feb. 2022). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 494–514. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3070843.

[37] Oliver Karras et al. “Divide and Conquer the EmpiRE: A Community-Maintainable Knowledge Graph of Empirical Research in Requirements Engineering”. In: *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Oct. 2023, pp. 1–12. DOI: 10.1109/ESEM56168.2023.10304795.

[38] Michael Färber. "The Microsoft Academic Knowledge Graph: A Linked Data Source with 8 Billion Triples of Scholarly Data". In: ed. by Chiara Ghidini et al. Vol. 11779. Cham: Springer International Publishing, 2019, pp. 113–129. ISBN: 978-3-030-30796-7. DOI: 10.1007/978-3-030-30796-7_8.

[39] Michael Färber and Lin Ao. "The Microsoft Academic Knowledge Graph enhanced: Author name disambiguation, publication classification, and embeddings". In: *Quantitative Science Studies* 3.1 (Apr. 12, 2022), pp. 51–98. ISSN: 2641-3337. DOI: 10.1162/qss_a_00183.

[40] Jason Priem, Heather Piwowar, and Richard Orr. *OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts*. June 17, 2022. DOI: 10.48550/arXiv.2205.01833. arXiv: 2205.01833[cs].

[41] Tony Hammond, Michele Pasin, and Evangelos Theodoridis. "Data integration and disintegration: Managing Springer Nature SciGraph with SHACL and OWL". In: *International Workshop on the Semantic Web*. 2017.

[42] Rodney Kinney et al. *The Semantic Scholar Open Data Platform*. Jan. 24, 2023. URL: http://arxiv.org/abs/2301.10140 (visited on 07/09/2024).

[43] Paolo Manghi et al. "The Data Model of the OpenAIRE Scientific Communication e-Infrastructure". In: *Metadata and Semantics Research*. Ed. by Juan Manuel Dodero, Manuel Palomo-Duarte, and Pythagoras Karampiperis. Berlin, Heidelberg: Springer, 2012, pp. 168–180. ISBN: 978-3-642-35233-1. DOI: 10.1007/978-3-642-35233-1_18.

[44] Amir Aryani and Jingbo Wang. "Research Graph: Building a Distributed Graph of Scholarly Works using Research Data Switchboard". In: Artwork Size: 505565 Bytes. Monash University, 2017. DOI: 10.4225/03/58C696655AF8A.

[45] Adrian Burton et al. "The Scholix Framework for Interoperability in Data-Literature Information Exchange". In: *D-Lib Magazine* 23.1 (Jan. 2017). ISSN: 1082-9873. DOI: 10.1045/january2017-burton.

[46] Daniel Domingo-Fernández et al. "COVID-19 Knowledge Graph: a computable, multi-modal, cause-and-effect knowledge model of COVID-19 pathophysiology". In: *Bioinformatics* 37.9 (June 9, 2021). Ed. by Lenore Cowen, pp. 1332–1334. ISSN: 1367-4803, 1367-4811. DOI: 10.1093/bioinformatics/btaa834.

[47] David Schindler, Benjamin Zapilko, and Frank Krüger. "Investigating Software Usage in the Social Sciences: A Knowledge Graph Approach". In: 12123 (2020). Ed. by Andreas Harth et al., pp. 271–286. DOI: 10.1007/978-3-030-49461-2_16.

[48] Danilo Dessí et al. "CS-KG: A Large-Scale Knowledge Graph of Research Entities and Claims in Computer Science". In: *The Semantic Web – ISWC 2022*. Ed. by Ulrike Sattler et al. Vol. 13489. Cham: Springer International Publishing, 2022, pp. 678–696. ISBN: 978-3-031-19433-7. URL: https://link.springer.com/10.1007/978-3-031-19433-7_39 (visited on 01/21/2025).

[49] Giuliana Spadaro et al. "The Cooperation Databank: Machine-Readable Science Accelerates Research Synthesis". In: *Perspectives on Psychological Science* 17.5 (Sept. 2022), pp. 1472–1489. ISSN: 1745-6916, 1745-6924. DOI: 10.1177/17456916211053319.

[50] Lyubomir Penev et al. “OpenBiodiv: A Knowledge Graph for Literature-Extracted Linked Open Data in Biodiversity Science”. In: *Publications* 7.2 (May 29, 2019), p. 38. ISSN: 2304-6775. DOI: 10.3390/publications7020038.

[51] Vinodh Ilangovan et al. *Open Research Knowledge Graph*. Cuvillier Verlag, July 5, 2024. 148 pp. ISBN: 978-3-68952-038-0.

[52] Holger Knublauch and Dimitris Kontokostas. *Shapes constraint language (SHACL)*. W3C, July 2017. URL: https://www.w3.org/TR/shacl/.

[53] Angelika Kaplan et al. “Combining Knowledge Graphs and Large Language Models to Ease Knowledge Access in Software Architecture Research”. In: *SemTech4STLD 2024 : Semantic Technologies and Deep Learning Models for Scientific, Technical and Legal Data 2024 : Second International Workshop on Semantic Technologies and Deep Learning Models for Scientific, Technical and Legal Data (SemTech4STLD) co-located with the Extended Semantic Web Conference 2024 (ESWC 2024). Ed.: R. Dessi*. 2024, p. 76. URL: https://publikationen.bibliothek.kit.edu/1000171637 (visited on 07/16/2024).

[54] Marta Sabou et al. “Survey on challenges of Question Answering in the Semantic Web”. In: *Semant. web* 8.6 (Jan. 1, 2017), pp. 895–920. ISSN: 1570-0844. DOI: 10.3233/SW-160247. URL: https://doi.org/10.3233/SW-160247 (visited on 05/02/2025).

[55] L. Hirschman and R. Gaizauskas. “Natural language question answering: the view from here”. In: *Nat. Lang. Eng.* 7.4 (Dec. 1, 2001), pp. 275–300. ISSN: 1351-3249. DOI: 10.1017/S1351324901002807. URL: https://doi.org/10.1017/S1351324901002807 (visited on 05/02/2025).

[56] Aidan Hogan et al. “Knowledge Graphs”. In: *ACM Computing Surveys* 54.4 (May 31, 2022), pp. 1–37. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3447772. arXiv: 2003.02320[cs].

[57] Bin Fu et al. *A Survey on Complex Question Answering over Knowledge Base: Recent Advances and Challenges*. July 26, 2020. DOI: 10.48550/arXiv.2007.13069. arXiv: 2007.13069.

[58] Hyung Won Chung et al. *Scaling Instruction-Finetuned Language Models*. Dec. 6, 2022. DOI: 10.48550/arXiv.2210.11416. arXiv: 2210.11416[cs].

[59] Takeshi Kojima et al. *Large Language Models are Zero-Shot Reasoners*. Jan. 29, 2023. URL: http://arxiv.org/abs/2205.11916 (visited on 07/09/2024).

[60] OpenAI et al. *GPT-4 Technical Report*. Mar. 4, 2024. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774[cs].

[61] Pengcheng He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. Oct. 6, 2021. DOI: 10.48550/arXiv.2006.03654. arXiv: 2006.03654[cs].

[62] Jason Wei et al. *Emergent Abilities of Large Language Models*. Oct. 26, 2022. DOI: 10.48550/arXiv.2206.07682.

[63] Tom B. Brown et al. *Language Models are Few-Shot Learners*. July 22, 2020. DOI: 10.48550/arXiv.2005.14165. arXiv: 2005.14165[cs].

[64] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. Jan. 10, 2023. DOI: 10.48550/arXiv.2201.11903. arXiv: 2201.11903[cs].

[65] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2014. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[66] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. Sept. 7, 2013. DOI: 10.48550/arXiv.1301.3781. arXiv: 1301.3781[cs].

[67] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. NAACL-HLT 2019. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[68] Shangyu Wu et al. *Retrieval-Augmented Generation for Natural Language Processing: A Survey*. July 19, 2024. DOI: 10.48550/arXiv.2407.13193. arXiv: 2407.13193[cs]. (Visited on 12/23/2024).

[69] Hao Yu et al. *Evaluation of Retrieval-Augmented Generation: A Survey*. July 3, 2024. URL: http://arxiv.org/abs/2405.07437 (visited on 07/09/2024).

[70] Matthijs Douze et al. *The Faiss library*. Sept. 6, 2024. DOI: 10.48550/arXiv.2401.08281. arXiv: 2401.08281[cs].

[71] Ashkan Alinejad, Krtin Kumar, and Ali Vahdat. *Evaluating the Retrieval Component in LLM-Based Question Answering Systems*. June 10, 2024. URL: http://arxiv.org/abs/2406.06458 (visited on 07/09/2024).

[72] Nourhan Ibrahim et al. "A survey on augmenting knowledge graphs (KGs) with large language models (LLMs): models, evaluation metrics, benchmarks, and challenges". In: *Discover Artificial Intelligence* 4.1 (Nov. 4, 2024), p. 76. ISSN: 2731-0809. DOI: 10.1007/s44163-024-00175-8.

[73] Taojun Hu and Xiao-Hua Zhou. *Unveiling LLM Evaluation Focused on Metrics: Challenges and Solutions*. version: 1. Apr. 14, 2024. DOI: 10.48550/arXiv.2404.09135. arXiv: 2404.09135[cs].

[74] Shahul Es et al. *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. Sept. 26, 2023. URL: http://arxiv.org/abs/2309.15217 (visited on 07/09/2024).

[75] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/ (visited on 05/27/2025).

[76] Kishore Papineni et al. “BLEU: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL ’02*. the 40th Annual Meeting. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2001, p. 311. DOI: 10.3115/1073083.1073135.

[77] Tianyi Zhang et al. *BERTScore: Evaluating Text Generation with BERT*. Feb. 24, 2020. URL: http://arxiv.org/abs/1904.09675 (visited on 07/09/2024).

[78] Angelika Kaplan et al. “Responsible and Sustainable AI: Considering Energy Consumption in Automated Text Classification Evaluation Tasks”. In: *Proceedings of the 2025 {IEEE/ACM} 47th International Conference on Software Engineering: Companion Proceedings, {ICSE} Companion 2025, Ottawa, 27th April - 3rd May 2025*. International Conference on Software Engineering (ICSE 2025), Ottawa, Kanada, 27.04.2025 – 03.05.2025. ISSN: 1558-1225. 2025. ISBN: 979-8-3315-0569-1. URL: https://publikationen.bibliothek.kit.edu/1000180194 (visited on 04/29/2025).

[79] Dennis Kundisch et al. “An Update for Taxonomy Designers”. In: *Business & Information Systems Engineering* 64.4 (Aug. 1, 2022), pp. 421–439. ISSN: 1867-0202. DOI: 10.1007/s12599-021-00723-x.

[80] Muhammad Usman et al. “Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method”. In: *Information and Software Technology* 85 (May 1, 2017), pp. 43–59. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2017.01.006.

[81] Robert C Nickerson, Upkar Varshney, and Jan Muntermann. “A method for taxonomy development and its application in information systems”. In: *European Journal of Information Systems* 22.3 (May 2013), pp. 336–359. ISSN: 0960-085X, 1476-9344. DOI: 10.1057/ejis.2012.26.

[82] Angelika Kaplan et al. “Introducing an Evaluation Method for Taxonomies”. In: *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*. EASE ’22. New York, NY, USA: Association for Computing Machinery, June 13, 2022, pp. 311–316. ISBN: 978-1-4503-9613-4. DOI: 10.1145/3530019.3535305.

[83] Kenneth D. Bailey. *Typologies and taxonomies: an introduction to classification techniques*. Nachdr. Sage university papers Quantitative applicatons in the social sciences 102. Thousand Oaks, Calif.: Sage Publ, 2003. 90 pp. ISBN: 978-0-8039-5259-1.

[84] Sussy Bayona-Oré et al. “Critical success factors taxonomy for software process deployment”. In: *Software Quality Journal* 22.1 (Mar. 1, 2014), pp. 21–48. ISSN: 1573-1367. DOI: 10.1007/s11219-012-9190-y.

[85] Lingxi Zhang et al. “A survey on complex factual question answering”. In: *AI Open* 4 (Jan. 1, 2023), pp. 1–12. ISSN: 2666-6510. DOI: 10.1016/j.aiopen.2022.12.003.

[86] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Sept. 19, 2023. DOI: 10.48550/arXiv.1910.10683. arXiv: 1910.10683[cs].

[87] Mingze Li et al. "KGMistral: Towards Boosting the Performance of Large Language Models for Question Answering with Knowledge Graph Integration". In: Workshop on Deep Learning and Large Language Models for Knowledge Graphs. July 9, 2024. URL: https://openreview.net/forum?id=JzL0qm3YA8 (visited on 11/20/2024).

[88] Jiashuo Sun et al. *Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph*. Mar. 24, 2024. URL: http://arxiv.org/abs/2307.07697 (visited on 07/09/2024).

[89] Chao Feng, Xinyu Zhang, and Zichu Fei. *Knowledge Solver: Teaching LLMs to Search for Domain Knowledge from Knowledge Graphs*. Sept. 6, 2023. DOI: 10.48550/arXiv.2309.03118. arXiv: 2309.03118.

[90] Lei Sun et al. *ODA: Observation-Driven Agent for integrating LLMs and Knowledge Graphs*. June 4, 2024. DOI: 10.48550/arXiv.2404.07677. arXiv: 2404.07677[cs]. URL: http://arxiv.org/abs/2404.07677 (visited on 04/14/2025).

[91] Bowen Jin et al. *Graph Chain-of-Thought: Augmenting Large Language Models by Reasoning on Graphs*. Oct. 3, 2024. DOI: 10.48550/arXiv.2404.07103. arXiv: 2404.07103[cs].

[92] Jinhao Jiang et al. *StructGPT: A General Framework for Large Language Model to Reason over Structured Data*. Oct. 23, 2023. URL: http://arxiv.org/abs/2305.09645 (visited on 07/09/2024).

[93] Yuan Sui et al. *FiDeLiS: Faithful Reasoning in Large Language Model for Knowledge Graph Question Answering*. Oct. 10, 2024. DOI: 10.48550/arXiv.2405.13873. arXiv: 2405.13873[cs].

[94] Shengjie Ma et al. *Think-on-Graph 2.0: Deep and Faithful Large Language Model Reasoning with Knowledge-guided Retrieval Augmented Generation*. Dec. 9, 2024. DOI: 10.48550/arXiv.2407.10805.

[95] Yao Xu et al. *Generate-on-Graph: Treat LLM as both Agent and KG in Incomplete Knowledge Graph Question Answering*. Oct. 6, 2024. DOI: 10.48550/arXiv.2404.14741.

[96] Yilin Wen, Zifeng Wang, and Jimeng Sun. *MindMap: Knowledge Graph Prompting Sparks Graph of Thoughts in Large Language Models*. Mar. 2, 2024. DOI: 10.48550/arXiv.2308.09729.

[97] Jiho Kim et al. *KG-GPT: A General Framework for Reasoning on Knowledge Graphs Using Large Language Models*. Oct. 17, 2023. DOI: 10.48550/arXiv.2310.11220.

[98] Yuqi Wang et al. *Reasoning on Efficient Knowledge Paths: Knowledge Graph Guides Large Language Model for Domain Question Answering*. Apr. 16, 2024. DOI: 10.48550/arXiv.2404.10384.

[99] Jinheon Baek et al. *Direct Fact Retrieval from Knowledge Graphs without Entity Linking*. May 21, 2023. DOI: 10.48550/arXiv.2305.12416.

[100] Jiahang Cao et al. "Knowledge Graph Embedding: A Survey from the Perspective of Representation Spaces". In: *ACM Computing Surveys* 56.6 (June 30, 2024), pp. 1–42. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3643806.

[101] Zhiyuan Zhang et al. “Pretrain-KGE: Learning Knowledge Representation from Pretrained Language Models”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Findings 2020. Ed. by Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, Nov. 2020, pp. 259–266. DOI: 10.18653/v1/2020.findings-emnlp.25.

[102] Xiaozhi Wang et al. *KEPLER: A Unified Model for Knowledge Embedding and Pre-trained Language Representation*. Nov. 23, 2020. DOI: 10.48550/arXiv.1911.06136. arXiv: 1911.06136[cs].

[103] Mojtaba Nayyeri et al. *Integrating Knowledge Graph embedding and pretrained Language Models in Hypercomplex Spaces*. Aug. 16, 2023. DOI: 10.48550/arXiv.2208.02743.

[104] Sören Auer et al. “The SciQA Scientific Question Answering Benchmark for Scholarly Knowledge”. In: *Scientific Reports* 13.1 (May 4, 2023), p. 7240. ISSN: 2045-2322. DOI: 10.1038/s41598-023-33607-z.

[105] Dan Moldovan et al. “The Structure and Performance of an Open-Domain Question Answering System”. In: *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*. ACL 2000. Hong Kong: Association for Computational Linguistics, Oct. 2000, pp. 563–570. DOI: 10.3115/1075218.1075289.

[106] J. T. Dillon. “The Classification of Research Questions”. In: *Review of Educational Research* 54.3 (Sept. 1, 1984), pp. 327–361. DOI: 10.3102/00346543054003327.

[107] Nguyen Thuan, Andreas Drechsler, and Pedro Antunes. “Construction of Design Science Research Questions”. In: *Communications of the Association for Information Systems* 44.1 (Mar. 1, 2019). ISSN: 1529-3181. DOI: 10.17705/1CAIS.04420.

[108] Kurt Bollacker et al. “Freebase: a collaboratively created graph database for structuring human knowledge”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD ’08. New York, NY, USA: Association for Computing Machinery, June 9, 2008, pp. 1247–1250. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376746.

[109] Sören Auer et al. “DBpedia: A Nucleus for a Web of Open Data”. In: *The Semantic Web*. Ed. by Karl Aberer et al. Berlin, Heidelberg: Springer, 2007, pp. 722–735. ISBN: 978-3-540-76298-0. DOI: 10.1007/978-3-540-76298-0_52.

[110] Denny Vrandečić and Markus Krötzsch. “Wikidata: a free collaborative knowledgebase”. In: *Commun. ACM* 57.10 (Sept. 23, 2014), pp. 78–85. ISSN: 0001-0782. DOI: 10.1145/2629489.

[111] Jonathan Berant et al. “Semantic Parsing on Freebase from Question-Answer Pairs”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2013. Ed. by David Yarowsky et al. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. URL: https://aclanthology.org/D13-1160 (visited on 09/14/2024).

[112] Qingqing Cai and Alexander Yates. "Large-scale Semantic Parsing via Schema Matching and Lexicon Extension". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2013. Ed. by Hinrich Schuetze, Pascale Fung, and Massimo Poesio. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 423–433. URL: https://aclanthology.org/P13-1042/ (visited on 04/17/2025).

[113] Wen-tau Yih et al. "The Value of Semantic Parse Labeling for Knowledge Base Question Answering". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. ACL 2016. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 201–206. DOI: 10.18653/v1/P16-2033.

[114] Antoine Bordes et al. *Large-scale Simple Question Answering with Memory Networks*. June 5, 2015. URL: http://arxiv.org/abs/1506.02075 (visited on 09/23/2024).

[115] Dennis Diefenbach et al. "Question Answering Benchmarks for Wikidata". In: *ISWC 2017*. Vienne, Austria, Oct. 2017. URL: https://hal.science/hal-01637141 (visited on 05/27/2025).

[116] Alon Talmor and Jonathan Berant. *The Web as a Knowledge-base for Answering Complex Questions*. Mar. 18, 2018. DOI: 10.48550/arXiv.1803.06643.

[117] Priyansh Trivedi et al. "LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs". In: *The Semantic Web – ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, Oct. 21, 2017, pp. 210–218. ISBN: 978-3-319-68203-7. DOI: 10.1007/978-3-319-68204-4_22.

[118] Ricardo Usbeck et al. "QALD-10 – The 10th challenge on question answering over linked data". In: *Semantic Web* Preprint (Preprint Jan. 1, 2023), pp. 1–15. ISSN: 1570-0844. DOI: 10.3233/SW-233471.

[119] Xueli Pan et al. *A RAG Approach for Generating Competency Questions in Ontology Engineering*. Sept. 13, 2024. DOI: 10.48550/arXiv.2409.08820.

[120] Yuntong Hu et al. *GRAG: Graph Retrieval-Augmented Generation*. May 26, 2024. URL: http://arxiv.org/abs/2405.16506 (visited on 07/09/2024).

[121] Victor R. Basili and David M. Weiss. "A Methodology for Collecting Valid Software Engineering Data". In: *IEEE Transactions on Software Engineering* SE-10.6 (Nov. 1984), pp. 728–738. ISSN: 1939-3520. DOI: 10.1109/TSE.1984.5010301.

[122] Victor R. Basili. *Software modeling and measurement: the Goal/Question/Metric paradigm*. Technical Report. Num Pages: 24. USA: University of Maryland at College Park, Aug. 1992.

[123] Claes Wohlin. "Guidelines for snowballing in systematic literature studies and a replication in software engineering". In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14. New York, NY, USA: Association for Computing Machinery, May 13, 2014, pp. 1–10. ISBN: 978-1-4503-2476-2. DOI: 10.1145/2601248.2601268. (Visited on 11/10/2024).

[124] Marco Schneider. *Replication Package of "HubLink: Leveraging Language Models for Enhanced Scholarly Information Retrieval on Research Knowledge Graphs"*. Karlsruhe, 2025. URL: https://gitlab.kit.edu/kit/kastel/sdq/stud/abschlussarbeiten/masterarbeiten/marco-schneider/ma-schneider-implementation.git.

[125] M. Shaw. "Writing good software engineering research papers". In: *25th International Conference on Software Engineering, 2003. Proceedings.* 25th International Conference on Software Engineering, 2003. Proceedings. May 2003, pp. 726–736. DOI: 10.1109/ICSE.2003.1201262.

[126] Steve Easterbrook et al. "Selecting Empirical Methods for Software Engineering Research". In: *Guide to Advanced Empirical Software Engineering*. Ed. by Forrest Shull, Janice Singer, and Dag I. K. Sjøberg. London: Springer, 2008, pp. 285–311. ISBN: 978-1-84800-044-5.

[127] Khiem Vinh Tran et al. *A Comparative Study of Question Answering over Knowledge Bases*. Nov. 15, 2022. DOI: 10.48550/arXiv.2211.08170.

[128] Tilahun Abedissa Taffa et al. *Hybrid-SQuAD: Hybrid Scholarly Question Answering Dataset*. Dec. 5, 2024. DOI: 10.48550/arXiv.2412.02788.

[129] Nadine Steinmetz and Kai-Uwe Sattler. "What is in the KGQA Benchmark Datasets? Survey on Challenges in Datasets for Question Answering on Knowledge Graphs". In: *Journal on Data Semantics* 10.3 (Dec. 1, 2021), pp. 241–265. ISSN: 1861-2040. DOI: 10.1007/s13740-021-00128-9.

[130] Valeriia Bolotova et al. "A Non-Factoid Question-Answering Taxonomy". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22. New York, NY, USA: Association for Computing Machinery, July 7, 2022, pp. 1196–1207. ISBN: 978-1-4503-8732-3. DOI: 10.1145/3477495.3531926.

[131] Zhe Liu and Bernard J. Jansen. "A Taxonomy for Classifying Questions Asked in Social Question and Answering". In: *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '15. New York, NY, USA: Association for Computing Machinery, Apr. 18, 2015, pp. 1947–1952. ISBN: 978-1-4503-3146-3. DOI: 10.1145/2702613.2732928.

[132] Ellen Riloff and Michael Thelen. "A Rule-based Question Answering System for Reading Comprehension Tests". In: *ANLP-NAACL 2000 Workshop: Reading Comprehension Tests as Evaluation for Computer-Based Language Understanding Systems*. 2000. DOI: 10.3115/1117595.1117598.

[133] Dat Quoc Nguyen and Son Bao Pham. "Ripple Down Rules for question answering". In: *Semantic Web* 8.4 (Jan. 1, 2017), pp. 511–532. ISSN: 1570-0844. DOI: 10.3233/SW-150204.

[134] Alexandr Chernov, Volha Petukhova, and Dietrich Klakow. "Linguistically Motivated Question Classification". In: *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*. NoDaLiDa 2015. Ed. by Beáta Megyesi. Vilnius, Lithuania: Linköping University Electronic Press, Sweden, May 2015, pp. 51–59. URL: https://aclanthology.org/W15-1809 (visited on 11/27/2024).

[135] SimmiK Ratan, Tanu Anand, and John Ratan. "Formulation of Research Question – Stepwise Approach". In: *Journal of Indian Association of Pediatric Surgeons* 24 (Jan. 1, 2019), p. 15. DOI: 10.4103/jiaps.JIAPS_76_18.

[136] Steven J. Kamper. "Types of Research Questions: Descriptive, Predictive, or Causal". In: *Journal of Orthopaedic & Sports Physical Therapy* 50.8 (Aug. 2020), pp. 468–469. ISSN: 0190-6011. DOI: 10.2519/jospt.2020.0703.

[137] Dag I. K. Sjoberg, Tore Dyba, and Magne Jorgensen. "The Future of Empirical Methods in Software Engineering Research". In: *Future of Software Engineering (FOSE '07)*. Future of Software Engineering (FOSE '07). May 2007, pp. 358–378. DOI: 10.1109/FOSE.2007.30.

[138] Ali Mohamed Nabil Allam and Mohamed H. Haggag. "The Question Answering Systems : A Survey ." In: 2016.

[139] Alexander Mikhailian, Tiphaine Dalmas, and Rani Pinchuk. "Learning foci for Question Answering over Topic Maps". In: *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*. ACL-IJCNLP 2009. Ed. by Keh-Yih Su et al. Suntec, Singapore: Association for Computational Linguistics, Aug. 2009, pp. 325–328. URL: https://aclanthology.org/P09-2082 (visited on 10/27/2024).

[140] Bilal Abu-Salih. "Domain-specific knowledge graphs: A survey". In: *Journal of Network and Computer Applications* 185 (July 2021), p. 103076. ISSN: 10848045. DOI: 10.1016/j.jnca.2021.103076.

[141] Marco Konersmann et al. "Evaluation Methods and Replicability of Software Architecture Research Objects". In: *2022 IEEE 19th International Conference on Software Architecture (ICSA)*. 2022 IEEE 19th International Conference on Software Architecture (ICSA). Honolulu, HI, USA: IEEE, Mar. 2022, pp. 157–168. ISBN: 978-1-6654-1728-0. DOI: 10.1109/ICSA53651.2022.00023.

[142] Yutong Zhao et al. "Butterfly Space: An Architectural Approach for Investigating Performance Issues". In: *2020 IEEE International Conference on Software Architecture (ICSA)*. 2020 IEEE International Conference on Software Architecture (ICSA). Mar. 2020, pp. 202–213. DOI: 10.1109/ICSA47634.2020.00027.

[143] Rebekka Wohlrab et al. "On Interfaces to Support Agile Architecting in Automotive: An Exploratory Case Study". In: *2019 IEEE International Conference on Software Architecture (ICSA)*. 2019 IEEE International Conference on Software Architecture (ICSA). Mar. 2019, pp. 161–170. DOI: 10.1109/ICSA.2019.00025.

[144] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (Apr. 1, 2009), pp. 131–164. ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8.

[145] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, Aug. 2, 1996, pp. 226–231. (Visited on 01/28/2025).

[146] Jenn Riley. *Understanding metadata: what is metadata, and what is it for*. NISO Primer series. Baltimore, MD: National Information Standards Organization, 2017. ISBN: 978-1-937522-72-8.

[147] Yixuan Tang and Yi Yang. *MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries*. Jan. 27, 2024. DOI: 10.48550/arXiv.2401.15391.

[148] Douglas C. Montgomery. *Design and Analysis of Experiments*. Google-Books-ID: Py7bDgAAQBAJ. John Wiley & Sons, 2017. 752 pp. ISBN: 978-1-119-11347-8.

[149] Wei-Lin Chiang et al. *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*. Mar. 7, 2024. DOI: 10.48550/arXiv.2403.04132.

[150] Kenneth Enevoldsen et al. *MMTEB: Massive Multilingual Text Embedding Benchmark*. Feb. 19, 2025. DOI: 10.48550/arXiv.2502.13595.

[151] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. Jan. 22, 2025. DOI: 10.48550/arXiv.2501.12948.

[152] Yu A. Malkov and D. A. Yashunin. *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*. Aug. 14, 2018. DOI: 10.48550/arXiv.1603.09320.

[153] Claes Wohlin et al. *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer, 2024. ISBN: 978-3-662-69306-3. DOI: 10.1007/978-3-662-69306-3.

[154] R. Feldt and Ana Magazinius. "Validity Threats in Empirical Software Engineering Research - An Initial Survey". In: International Conference on Software Engineering and Knowledge Engineering. 2010.

[155] Mohammad Yani and Adila Alfa Krisnadhi. "Challenges, Techniques, and Trends of Simple Knowledge Graph Question Answering: A Survey". In: *Information* 12.7 (July 2021), p. 271. ISSN: 2078-2489. DOI: 10.3390/info12070271.

[156] Xiaoxiao Jin et al. "Floating-Point Embedding: Enhancing the Mathematical Comprehension of Large Language Models". In: *Symmetry* 16.4 (Apr. 2024), p. 478. ISSN: 2073-8994. DOI: 10.3390/sym16040478.

[157] Darren Edge et al. *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. Apr. 24, 2024. URL: http://arxiv.org/abs/2404.16130 (visited on 07/09/2024).

[158] Zirui Guo et al. *LightRAG: Simple and Fast Retrieval-Augmented Generation*. Nov. 7, 2024. DOI: 10.48550/arXiv.2410.05779.

A. Appendix

A.1. Prompts

A.1.1. HubLink

Listing A.1: Partial Answer Generation Prompt

```
### Task Description:
  You are given a question and a set of context passages. Your task is:
  1. Carefully examine the provided contexts.
  2. Extract all explicitly stated information from the contexts that helps answer
  the question (even if the information is partial, incomplete, or does not fully
  resolve the question).
  2. If you find relevant but partial information, you must clearly present it.
  3. ONLY IF no explicitly relevant information at all is present, then clearly
  state:
  ‘‘‘
  Insufficient Information.
  ‘‘‘
  ### Examples:
  #### Example 1: No Relevant Information
  - **Question:** "Who wrote Paper X?"
  - **Provided Texts:**
    "Paper X discusses software design."
  - **Answer:**
    ‘‘‘
    Insufficient information.
    ‘‘‘
  #### Example 2: Direct Information
  - **Question:** "When was Paper X published?"
  - **Provided Texts:**
    "Paper X was published in 2020."
  - **Answer:**
    "Paper X was published in 2020."
  #### Example 3: Partial Information
  - **Question:** "When was Paper Y published compared to Paper Z?"
  - **Provided Texts:**
    "Paper Y was published in 2018."
  - **Answer:**
```

```
  "Paper Y was published in 2018. However, the publication date of Paper Z is not
  provided in the texts. "
#### Example 4: Partial Information
- **Question:** "How many papers include X?"
- **Provided Texts:**
  "Paper A includes X."
- **Answer:**
  "The Paper with the title 'Paper A' includes X. However, this is only one
  instance, and there may be more papers that include X."
---
### Start Your Task
If any relevant information is found, even if it is just a single mention or a
partial detail, forward that information. Do not conclude 'Insufficient
Information' if at least one explicit data point exists.
**Question:**
{question}
**All Texts have the following information in common:**
{common_information}
**Provided Texts:**
{texts}
**Answer:**
```

Listing A.2: Final Answer Generation Prompt

```
### Task Description:
You are an expert text synthesizer. Your task is to combine the provided partial
answers into a coherent and comprehensive final answer. Each partial answer has a
source identification provided as [ID] that you should cite in the final answer.

### Constraints:

- Do **not** introduce any external information, personal opinions, or
interpretations beyond what is provided in the partial answers.
- If a partial answer contains identification objects such as object_1, object_2,
etc., do not include the object identification in the final aswer just use the
source id [ID] as mentioned above.
- Ensure clarity and coherence in the final output.

### Question:
{question}

### Partial Answers:
{partial_answers}

### Final Answer:
```

Listing A.3: Question Component Extraction Prompt

```
### Task Description:
```

```
Please analyze the following question and break it down into its key components or
claims.

### Instructions:
- Break down the question into its key components or claims.
- Your output has to be a valid python list of strings.
- Do not use any external information, personal opinions, or interpretations
beyond the given texts. Do not make any assumptions and do not answer the question.
- Do not include any additional explanations or commentary. Just provide the list
of components.

### Examples:
Question: What are the research questions in the paper titled 'Artificial
Intelligence in Healthcare'?
Components: ['Research Questions', 'Artificial Intelligence in Healthcare']
Question: Which publications include the research object 'Architecture Analysis
Method'?
Components: ['Research Object', 'Architecture Analysis Method']
Question: Which papers have been published by SpringerLink in 2020?
Components: ['Publisher', 'SpringerLink', '2020']

### Start Your Task:
Question: {question}
```

Listing A.4: Triple Filtering Prompt

```
### Task Description:
You are a knowledgeable assistant. Given a question, a answer and a list of
ID-triple mappings in the format "ID: (subject, predicate, object)" determine
which triples provide direct information essential for answering the question. If
a triple is relevant or partially relevant, include its ID in the final output;
otherwise, exclude it. Provide in your answer only a list of the final triple IDs
and do not provide any additional explanations or commentary. If no triples are
relevant, return an empty list.

### Required Output Format:
[ID1, ID2, ID3, ...]

### Question:
{question}

### Answer:
{answer}

### Retrieved Contexts:
{contexts}
```

A.1.2. Question Augmentation and Reranking

Listing A.5: Question Augmentation Prompt

```
You are an assistant designed to enhance user questions to improve the effectiveness
of a Retrieval-Augmented Generation (RAG) system. Your task is to augment the
original question by clarifying ambiguities, incorporating related keywords or
phrases that will help the retrieval system fetch more accurate and comprehensive
information, and add nouns or noun phrases to terms to clearly indicate their types
or roles.

**Guidelines:**
1. **Clarify Ambiguous Terms:** If the original question contains terms that could
be interpreted in multiple ways, clarify their meanings based on common usage or
provide additional context.
2. **Incorporate Related Keywords:** Add synonyms or related terms that are
pertinent to the topic to broaden the retrieval scope without deviating from the
original intent.
3. **Explicit Noun Phrases:** Incorporate nouns or noun phrases to terms to specify
their types (e.g., author, year, title, person, etc.)
4. **Maintain Original Intent:** Ensure that the augmented question retains the
original purpose and does not introduce unrelated elements.

**Format:**
Provide the augmented question directly without additional commentary.

**Original Question:** "{original_question}"
```

Listing A.6: Context Reranking Prompt

```
You are an intelligent assistant specialized in information retrieval and ranking.

Given a user question and a list of retrieved context IDs mapped to their respective
contexts, your task is to rerank the context IDs in order of their relevance to the
question. The structure of the mapping is 'ID: context'. The most relevant context
ID should be listed first, followed by the next most relevant, and so on.
Think in steps: First determine the most important entity that is asked for in the
question, then check which contexts are the most relevant to that entity. Then
continue to the next most important entity and so on.

Provide only the "Re-Ranked Context IDs" for example: [2, 5, 8] without any
additional explanations or commentary.

**Question:**
{question}

**Retrieved Contexts:**
{contexts}
```

```
**Instructions:**
- Analyze each context in relation to the question.
- Determine the relevance of each context to answering the question.
- Rank the context IDs from most relevant (1) to least relevant (n).

**Re-Ranked Contexts:**
```

Listing A.7: Answer Generation Prompt

```
You are an advanced AI language model tasked with answering questions based solely
on the provided context.
Your responses must be accurate, concise, and derived exclusively from the given
information.

## Context Explanation:
{context_explanation}

## Context:
{context}

## Question:
{question}

## Instructions:
1. Carefully analyze the provided context.
2. Formulate a clear and concise answer based exclusively on the given context.
3. If the context lacks sufficient information to answer the question fully:
a. Provide any partial information that is relevant and available.
b. Clearly state: "The question cannot be fully answered with the provided context."
4. Do not introduce any external information or make assumptions beyond the given
context.
5. If dealing with vector store chunks, try to connect information across chunks
when relevant.
6. For knowledge graph data, interpret the relationships between entities to derive
your answer.
7. Maintain a confident tone, but also indicate any uncertainties that arise from
limitations in the context.

## Answer:
```

A.2. ORKG Contribution Templates

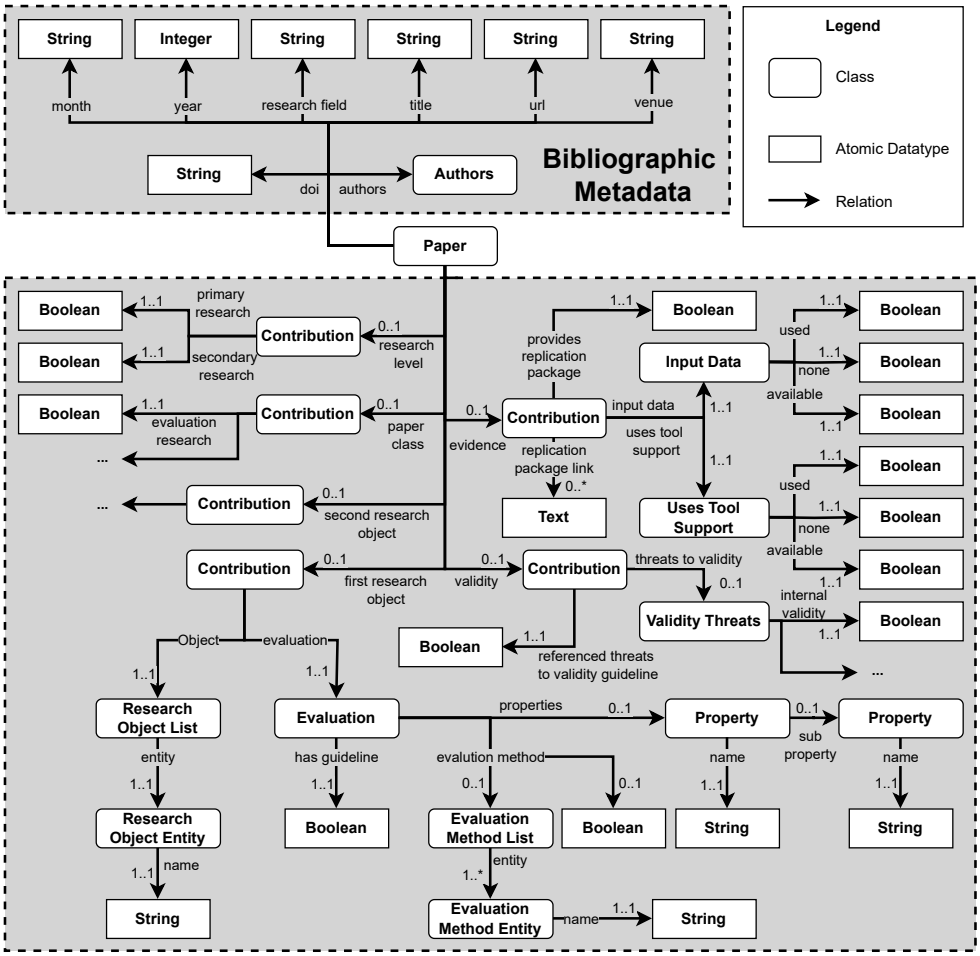


Figure A.1.: Template for the **GV1** graph variant which stores data deep and distributed.

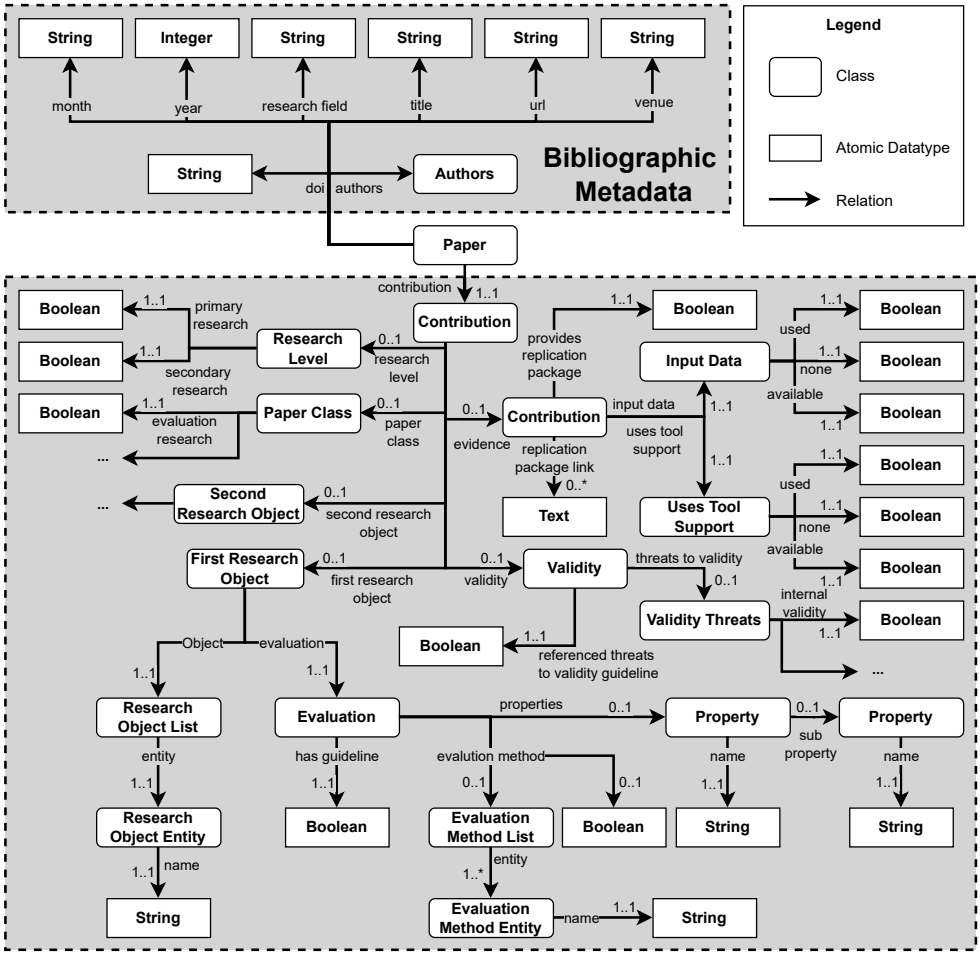


Figure A.2.: Template for the **GV2** graph variant which stores data deep and centralized.

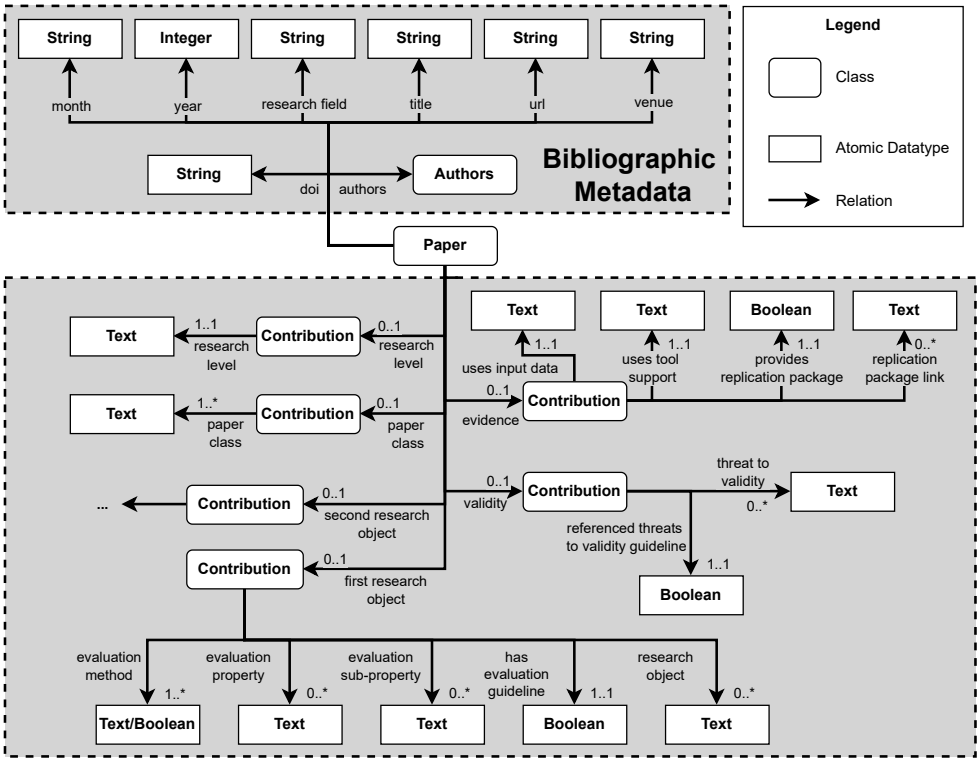


Figure A.3.: Template for the **GV3** graph variant which stores data flat and distributed.

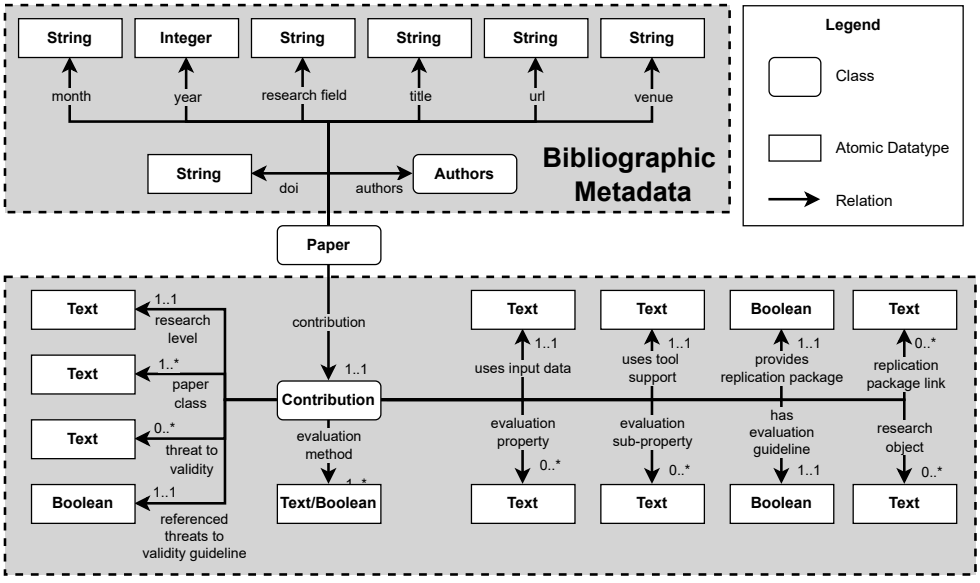


Figure A.4.: Template for the **GV4** graph variant which stores data flat and centralized.

A.3. Additional Evaluation Results

A.3.1. Additional Evaluation Results for Operation Complexity

Retrieval Operation	Recall	Precision	F1	Hits@10	Map@10	MRR@10	EM@10
			DiFaR				
basic	0.528	0.006	0.006	0.167	0.083	0.083	0.017
aggregation	0.365	0.008	0.017	0.236	0.199	0.302	0.083
counting	0.350	0.007	0.017	0.257	0.215	0.267	0.092
ranking	0.164	0.005	0.013	0.111	0.089	0.190	0.058
comparative	0.363	0.014	0.029	0.247	0.130	0.331	0.154
relationship	0.380	0.017	0.032	0.270	0.239	0.545	0.196
negation	0.364	0.013	0.031	0.219	0.107	0.308	0.131
superlative	0.346	0.018	0.034	0.113	0.087	0.278	0.075
			Mindmap				
basic	0.278	0.020	0.037	0.056	0.006	0.006	0.006
aggregation	0.115	0.031	0.046	0.004	0.000	0.004	0.004
counting	0.182	0.041	0.065	0.004	0.000	0.004	0.004
ranking	0.105	0.029	0.043	0.016	0.002	0.009	0.008
comparative	0.026	0.010	0.014	0.005	0.005	0.042	0.004
relationship	0.116	0.055	0.073	0.018	0.002	0.015	0.013
negation	0.054	0.020	0.029	0.023	0.003	0.023	0.019
superlative	0.079	0.030	0.041	0.000	0.000	0.000	0.000
			FiDeLiS				
basic	0.333	0.208	0.245	0.333	0.306	0.306	0.220
aggregation	0.035	0.016	0.022	0.035	0.013	0.028	0.017
counting	0.021	0.012	0.015	0.021	0.010	0.021	0.012
ranking	0.102	0.037	0.051	0.101	0.038	0.072	0.029
comparative	0.157	0.067	0.087	0.157	0.099	0.173	0.079
relationship	0.052	0.044	0.048	0.053	0.033	0.121	0.046
negation	0.011	0.011	0.011	0.010	0.010	0.062	0.006
superlative	0.045	0.047	0.045	0.045	0.019	0.062	0.031

Table A.1.: Impact of the retrieval operation on the performance of the KGQA baseline approaches. The results are based on graph variant **GV1** and all metrics have been macro averaged.

A.3.2. Additional Evaluation Results for Use Cases

Use Case	Recall	Precision	F1	Hits@10	Map@10	MRR@10	EM@10
DiFaR							
1	0.588	0.010	0.021	0.436	0.352	0.538	0.129
2	0.104	0.001	0.001	0.000	0.000	0.000	0.000
3	0.453	0.016	0.032	0.302	0.238	0.412	0.156
4	0.264	0.011	0.021	0.191	0.145	0.324	0.121
5	0.275	0.009	0.020	0.070	0.029	0.101	0.048
6	0.421	0.014	0.031	0.255	0.152	0.400	0.155
Mindmap							
1	0.177	0.034	0.053	0.005	0.001	0.006	0.004
2	0.023	0.014	0.017	0.000	0.000	0.000	0.000
3	0.208	0.033	0.054	0.035	0.004	0.007	0.007
4	0.121	0.043	0.061	0.022	0.002	0.015	0.014
5	0.098	0.034	0.048	0.016	0.005	0.041	0.012
6	0.072	0.020	0.030	0.004	0.000	0.003	0.003
FiDeLiS							
1	0.269	0.253	0.271	0.148	0.268	0.183	0.182
2	0.125	0.043	0.065	0.036	0.125	0.055	0.025
3	0.057	0.036	0.060	0.052	0.057	0.052	0.047
4	0.061	0.036	0.150	0.053	0.061	0.056	0.043
5	0.040	0.018	0.033	0.014	0.041	0.020	0.012
6	0.047	0.025	0.076	0.030	0.047	0.035	0.031

Table A.2.: Assessment of different scholarly use cases on the retrieval performance for the baseline KGQA approaches. The results are based on graph variant **GV1** and all metrics have been macro averaged. The use cases are introduced in Section 8.3.1.

A.3.3. Additional Evaluation Results for Type Information in the Question

Semi-Typed	Recall	Precision	F1	Hits@10	Map@10	MRR@10	EM@10
			DiFaR				
True	0.256	0.189	0.349	0.011	0.359	0.023	0.128
False	0.155	0.109	0.238	0.010	0.343	0.021	0.078
			Mindmap				
True	0.006	0.001	0.006	0.029	0.105	0.042	0.005
False	0.024	0.004	0.021	0.031	0.133	0.047	0.010
			FiDeLiS				
True	0.108	0.076	0.121	0.065	0.108	0.076	0.062
False	0.075	0.048	0.083	0.039	0.075	0.049	0.043

Table A.3.: The impact of questions that add information about the condition types compared to those that do not for the baseline KGQA approaches. The results are based on graph variant **GV1** and all metrics have been macro averaged.