

Unveiling Hidden Information in Unstructured Documents: Organization and Hybrid Retrieval with Knowledge Graphs.

Supervisor: Prof. Matteo Luigi Palmonari

Co-supervisor: Dott. Francesco Abbracciavento

Co-supervisor: Dott. Riccardo Pozzi

Master's degree thesis by:

Davide Giardini

ID number 897473

Academic Year 2023-2024

Contents

1	Introduction	1
1.1	Objectives	4
1.1.1	Efficient Document Structuring	5
1.1.2	Hybrid Retrieval System for QA	7
1.1.3	Comparison with traditional RAG paradigms	10
1.2	Key Research Questions and Thesis Outline	10
2	Related Work on Question Answering Systems	12
2.1	Overview of QA systems	13
2.2	Large Language Models for QA	14
2.3	Hybrid QA Models	16
2.4	Retrieval Augmented Generation for QA	17
2.5	Knowledge Graphs for QA	20
2.6	Document structuring for QA	23
2.7	Improving Retrieval in KGQA	29
2.7.1	Node Retrieval	30
2.7.2	Search Module	31
2.7.3	Subgraph Retrieval	33
2.7.4	Hybrid Retrieval	34
2.8	Summary of Findings and Key Open Challenges	36
3	Methodology	40
3.1	Document Structuring	41
3.2	Retrieval	46
3.2.1	Retrieval Approaches	48
3.2.2	Hybrid Retrieval	54
4	Experiments	59
4.1	Dataset	59
4.1.1	Natural Questions Dataset	59
4.1.2	Synthetic Multi-Hop Dataset	64

4.2	Baselines	67
4.3	Evaluation of the Retrieval System	68
4.3.1	Retrieval Metrics	69
4.3.2	Retrieval Performance	71
4.4	Evaluation of the RAG System	75
4.4.1	RAG Metrics	76
4.4.2	RAG Performance	77
4.5	Additional Evaluations and Findings	77
4.5.1	Features Importance in Retrieval	78
4.5.2	Evaluation of Key Methodological Choices	80
4.5.3	Performance with Limited Training Data	80
5	Conclusion and Future Developments	82
List of Figures		97
List of Tables		98

Chapter 1

Introduction

The emergence of Large Language Models (LLMs) like OpenAI’s ChatGPT[1] in late 2022 marked a significant turning point in the evolution of artificial intelligence and its integration into everyday life. Since then, numerous other conversational agents have emerged, including Claude by Anthropic¹, Gemini by Google DeepMind[2], and several open-source alternatives, each contributing to the rapid expansion of the AI ecosystem. These AI-driven conversational agents have quickly demonstrated their potential to revolutionize a wide range of domains, from customer service and education to content creation and software development. By providing instant, coherent, and contextually relevant responses, they have become indispensable tools for both individuals and organizations, enhancing productivity and democratizing access to information. To perform tasks and answer users’ questions, neural language models exploit the knowledge they acquired during pre-training and that is now “stored” inside their parameters. However, this way of storing knowledge is not optimal and is the main cause of LLMs’ major limitations. One of the most pressing issues is their tendency to produce “hallucinations”: confidently presenting incorrect or fabricated information as factual. Additionally, these models operate within the constraints of a knowledge cutoff, meaning they cannot access or incorporate real-time information beyond their last training update. This results in outdated responses for rapidly evolving fields and an inability to provide personalized or context-specific information that falls outside their training corpus, such as private or highly specialized data. These problems constitute an incredible bottleneck in the implementation of AI in all of those tasks that are knowledge-intensive or in which the accuracy of the response is crucial.

This problem has been thoroughly tackled in the realm of Question An-

¹anthropic.com/news/introducing-claude

swering (QA), in which a retrieval system is typically employed to identify relevant documents that may contain answers to user queries. Once a suitable document is retrieved, further processing is conducted to extract the precise answer. Recent advancements have increasingly incorporated LLMs for this refinement step, demonstrating that their combination with a retrieval engine brings to the resolution of most of the model’s aforementioned problems.

This approach was later expanded and generalized by Lewis et al.[3] to the entire task of Sequence-to-Sequence (Seq2Seq) modelling with the introduction of Retrieval Augmented Generation (RAG). Instead of relying solely on their internal parametric memory, RAG models dynamically query external databases or document collections to supplement their responses with real-time, factual information. Notable implementations include AI agents like ChatGPT² and Copilot³, which can now search the Internet in real-time to supplement their knowledge with up-to-date information. Given all these benefits, RAG systems too have become a focal point of both academic research and practical deployment in industries[4], as they can make the difference between a helpful and a useless conversational agent. Key areas of investigation include the development of more efficient indexing methods[5][6][7], advanced re-ranking algorithms⁴⁵, improved Knowledge Base (KB) structures for optimized retrieval[8][9], and smarter strategies for integrating extracted context into the model to ensure more accurate and contextually aligned responses[10]. These efforts are further supported by ongoing work on the application of RAG systems to domain-specific applications, such as legal research[11], healthcare[12], and customer support[13], where precision and reliability are critical.

A robust retrieval engine is therefore crucial in enhancing the quality of outputs produced by an LLM, regardless of whether the task is QA or RAG. This component ensures that the model is provided with the most relevant and accurate context, thereby improving its overall performance and the reliability of the generated responses.

However, the majority of human knowledge, whether public, private, or internal to an organization, remains unstructured, typically existing in the form of textual documents[14][15]. To enable efficient retrieval, such data must first be organized within a structured repository accessible to the system. One of the most widely adopted approaches for this purpose is the

²openai.com/index/chatgpt/

³blogs.microsoft.com/blog/2023/09/21/announcing-microsoft-copilot-your-everyday-ai-companion/

⁴haystack.deepset.ai/blog/enhancing-rag-pipelines-in-haystack

⁵docs.haystack.deepset.ai/docs/lostinthemiddleranker

use of vector databases[16]. In them, each chunk of text is embedded into a semantically rich vector, which is then retrieved based on its proximity to the vector generated from the query, allowing the storage of unstructured documents in an easy and efficient manner. While this solution definitely has its advantages, the most important of which being the ease with which anyone can implement a functioning and useful QA or RAG solution, it focuses only on the content of the documents, leaving behind all the other information that can be inferred from them and that could prove crucial for a more efficient retrieval. An example of this is the organization of text within documents, which might be lost when stored in a vector database. Humans often rely on the inherent structure of documents, such as headings, paragraphs, and sections, to quickly locate the information of interest. This contextual organization aids in navigation and comprehension, and for this reason its removal deprives the retrieval system of crucial information needed to identify the most relevant chunk, thereby diminishing its effectiveness. Another crucial aspect is the presence of entity mentions and the semantic relationships between those entities, both within and across documents. While these connections are implicitly expressed in textual passages, a retrieval system that relies solely on raw text is unlikely to fully leverage them. This limitation is particularly evident when considering that the relationship between two entities is typically articulated within a single chunk of text and, as a result, is lost in any other mention of the same entities appearing in different paragraphs. There is a need for a structure that makes these relations explicit and suitable to be leveraged by the retrieval algorithm.

1.1 Objectives

We generalize the informational content of a document along three distinct dimensions:

1. **Content:** The textual content of the document.
2. **Structure:** The hierarchical organization of the document, including headings, paragraphs, and sections.
3. **Entities:** The entities mentioned within the documents and their semantic relations.

Given this categorization, our objective is to design a retrieval system, ranging from document structuring to the retrieval model itself, that can determine the most relevant chunks by integrating and combining *all* these informational dimensions.

Specifically, the **main objectives** of this thesis are the following:

1. **Document Structuring:** Developing a Knowledge Graph (KG) representation that effectively models a collection of unstructured documents, focusing on retaining and making explicit *all* of their informational dimensions: both the textual content and the more implicit metadata, such as the organization of passages and the relationships between entities.
2. **Hybrid Retrieval:** Designing a retrieval system capable of leveraging this enriched structure, and therefore taking full advantage of all three informational dimensions to retrieve the most relevant chunks comprehensively.
 - (a) Investigating which retrieval methods are more critical in identifying the most relevant passages, and consequently design the system to balance the weights in favor of those methods.
3. **Comparison with traditional RAG:** Understanding whether structuring and enriching documents can offer a valuable benefit in RAG systems.

The following sections will describe them in more detail.

1.1.1 Efficient Document Structuring

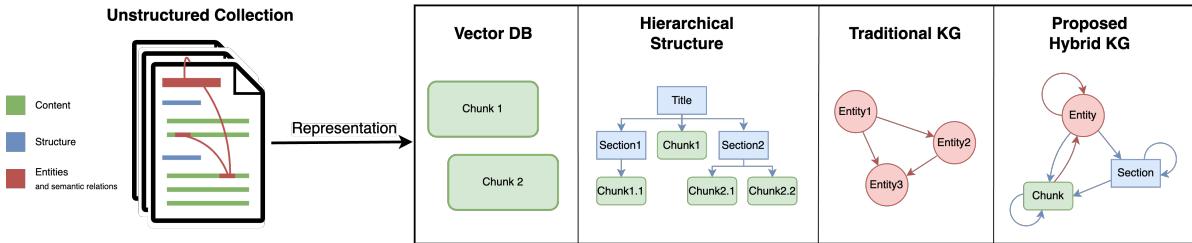


Figure 1.1: Comparison of different document representations

Vector DB maintains only the content of the document. A Hierarchical Structure maintains the document’s organization and its content. Traditional KGs maintain only entities and their semantic relations. Our proposed representation aims to maintain all the three different document’s informational dimensions.

In the first place, this thesis aims to create a KG structure that can effectively represent the initial set of unstructured documents in a new form, focusing on retaining the textual content while unveiling the more implicit information, such as the organization of the text and the relationships between the entities mentioned within it.

As discussed in greater detail in the chapter concerned with the state of the art (Section 2.6), existing document representations typically capture only one or a few informational dimensions of the original documents. Vector databases, for instance, structure documents as discrete chunks represented by their embeddings, preserving only textual content while disregarding structural and relational aspects. Hierarchical representations, on the other hand, focus on document organization but fail to capture entity information and semantic relationships. Similarly, a traditional KG, with its subject-predicate-object structure, often loses the structural dimension of the document and may omit some of its content. In contrast, our approach seeks to define a KG representation that explicitly preserves all three informational dimensions (textual, structural, and semantic), enabling a retrieval system to leverage all of them for the identification of relevant chunks.

We are interested in applying this structuring to *encyclopedic documents*. We define encyclopedic documents as any collection in which the user’s abstraction of an “entity” directly corresponds to the concept of a document itself. In simpler terms, this method is applicable to datasets where each document is dedicated to a single entity, providing a comprehensive definition and description of it. However, while each document must represent a

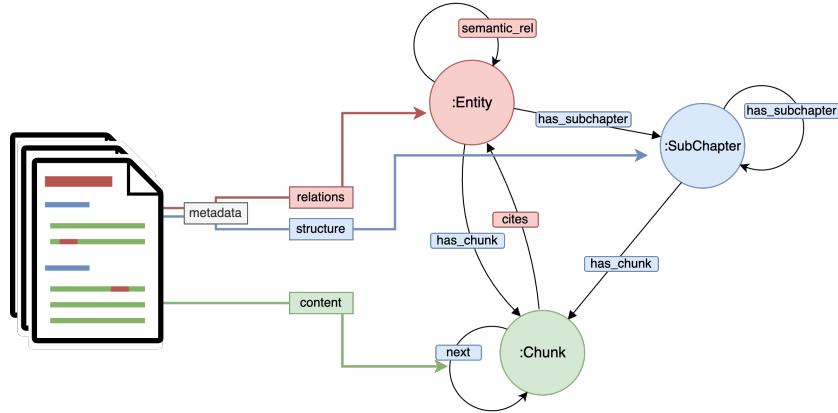


Figure 1.2: Documents structuring into the Knowledge Base

The content of the page is represented within the KG through “Chunk” nodes (depicted in green), which encapsulate segments of the text. The document’s structure and organization are explicitly modeled using “SubChapter” nodes and positional relationships (shown in blue). The central entity described by the document, along with all referenced entities, are stored in “Entity” nodes (colored in red) and linked by their semantic relations.

specific entity, our approach also allows for the inclusion of external entities that do not have a corresponding descriptive document within the collection. This means that during the entity extraction process, it is possible to identify and enrich our KB with additional entities that were not originally part of the document collection.

In particular, a parser is built to extract information from Wikipedia pages, and structure them in a KG stored in Neo4j⁶. We select Wikipedia as the knowledge source because it significantly simplifies the implementation of the experiment. First of all, Wikipedia follows the necessary encyclopedic structure: each page deals only with one main topic and each topic can be defined by its corresponding page. Furthermore, references to the other pages are highlighted with hyperlinks, making it easy to be parsed. In fact, while our approach is generalizable to every knowledge source with an encyclopedic structure, implementing it on plain text would require the beforehand application of Named Entity Recognition (NER)[17], Named Entity Linking (NEL) and Coreference Resolution algorithms[18]. Since the entities within a Wikipedia page are already highlighted and linked, we are able to focus on the main objective of the experiment (developing the structure of the KB and the retrieval system) by skipping the initial document preparation. The

⁶neo4j.com/

same can be said for the relations between entities: since parallel knowledge sources like WikiData or DBpedia exists, we can use them to retrieve the type of connection that links two entities, instead of extracting it with Relation Extraction models[19].

To achieve this goal, we design a KG whose schema is tailored to accommodate each information type. Entities, represented by the title of the corresponding page, are stored in “Entity” nodes. These nodes are interconnected by extracting the relative relations from DBpedia, but only when both entities co-occur within the same document. The document’s structure and organization is retained thanks to “SubChapter” nodes, which are linked back to their parent Entity node. These nodes facilitate content navigation by guiding the retrieval algorithm to the most relevant section of the page that addresses the user’s query. Finally, the actual page content is sliced in chunks, each forming a “Chunk” node. Chunk nodes are linked back to their parent Entity or SubChapter, as well as to their adjacent chunks and to all the entities they reference. A detailed overview of this approach is presented in figure 1.2.

1.1.2 Hybrid Retrieval System for QA

	Structure	Entities	Content
Entity Based Question: Where was the song "New Gold", in which Tame Impala is featured, performed live for the first time?	<p>Tame Impala#Tours</p> <p>InnerSpeaker Tour (2010-2011) Lonerism Tour (2012-2014)</p> <p>New Gold (song)#Background</p> <p>It was first played live during the Gorillaz World Tour 2022 on 19 August 2022.</p>	<p>Gorillaz#2022-present</p> <p>the band performed the new song "New Gold" (featuring Tame Impala) at All Points East in London</p> <p>New Gold (song)</p> <p>New Gold is a song by British band Gorillaz, featuring Australian music project Tame Impala, and...</p>	<p>Tame Impala#2018</p> <p>...2018 Mad Cool Festival in Spain, the first live music show the band agreed to play in 2018.</p> <p>the band performed the new song "New Gold" (featuring Tame Impala) at All Points East in London</p> <p>New Gold is a song by british band Gorillaz, featuring Australian music project Tame Impala, and...</p>
Broad Question: What is the main theme of Harry Potter?	<p>Harry Potter#Themes#1</p> <p>Harry Potter's overarching theme is death. In the first book, when Harry looks in the Mirror of Erised, he...</p> <p>Harry Potter#Themes#2</p> <p>Love distinguishes Harry and Voldemort. Harry is a hero because he loves others, even willing to...</p>	<p>Harry Potter</p> <p>Harry Potter is a series of seven fantasy novels written by the British author J.K. Rowling.</p> <p>J.K. Rowling</p> <p>...she is the author of Harry Potter, a seven-volume fantasy novel series published from 1997 to...</p>	<p>Harry Potter#Plot</p> <p>The series follows the life of a boy named Harry Potter. In the first book, Harry Potter and the...</p> <p>Harry Potter is a series of seven fantasy novels written by the British author J.K. Rowling.</p> <p>Harry Potter's overarching theme is death. In the first book, when Harry looks in the Mirror of Erised, he...</p>

Figure 1.3: Comparison of Retrieval Systems leveraging informational dimensions individually

The subsequent goal is to design a retrieval system capable of leveraging this enriched structure, and therefore taking full advantage of all of the document’s informational dimensions.

As illustrated by the two examples in Figure 1.3, we argue that a retrieval method leveraging only one or two informational dimensions of the original document is insufficient to appropriately handle all types of queries. For instance, the question “Where was the song New Gold, in which Tame Impala is featured, performed live for the first time?” is highly dependent on the entities mentioned. A retrieval system that relies exclusively on the hierarchical structure of the original document may struggle to locate the relevant chunks, as it does not explicitly consider entity relationships. Conversely, a system focused purely on content might not always retrieve the correct chunks. This issue arises because, when encoding the initial chunks into embeddings, the representation must be generic enough to retain all information. As a result, the mention of a specific entity may be diluted by other details present in the chunk, reducing its prominence in the resulting embedding. To maximize the probability of retrieving the correct chunks for these particular types of questions, a retrieval method must preserve explicitly mentioned entities in its representation of the original document and leverage them during retrieval by comparing them with the query.

On the other hand, a question such as “What is the main theme of Harry Potter?” directly refers to a specific section of the original document. A retrieval system that disregards the document’s hierarchical organization, whether to benefit entities or content’s representation, will struggle to extract the most relevant chunks for answering this query.

This highlights the need for a hybrid approach that can effectively integrate multiple informational dimensions to enhance retrieval quality across diverse question types.

We address this challenge with a Hybrid Retrieval system that amalgamates together relevance information from various retrieval methods, each focusing on a particular facet of the KG. Specifically, a “Section Retrieval”, a “Page Retrieval” and a “SubChapter Selector”, will measure a chunk’s relevance based on its position within the document’s structure, effectively leveraging the information about the document organization that has been unveiled by our KG structure. The perspective centered on entities and their interconnections is instead provided by a “SubGraph Retrieval” and an “Entity Retrieval” approach. Additionally, a “Vector Retrieval” method is tasked to retrieve the most relevant chunk purely based on its textual content. Finally, a Neural Network is trained to integrate these diverse sources, ultimately determining which Chunk nodes to retrieve to answer the user’s query.

The intuition behind our approach draws inspiration from ensemble models[20][21]: machine learning techniques that combine multiple individual models to enhance predictive performance beyond what a single model could

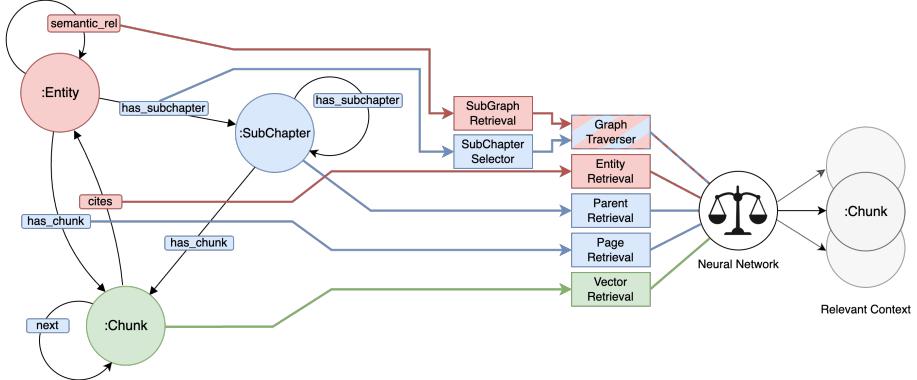


Figure 1.4: Retrieval Pipeline

Each retrieval method offers insights into a chunk’s relevance from its unique viewpoint: the entities’ semantic relationships (in red), the document’s organization (in blue) and the textual content (in green). They are then combined into a Neural Network to provide a final comprehensive measure of relevance.

achieve. These techniques harness the principle of collective intelligence, where the diversity and complementary strengths of multiple models are aggregated to reduce errors and improve accuracy. Similarly, our retrieval framework operates under the premise that each retrieval method offers insights into a chunk’s relevance from its unique viewpoint, and the Neural Network effectively integrates these insights to deliver a final, optimized retrieval decision.

Weighted Hybrid Retrieval

This leads to a secondary research question in this thesis: understanding which retrieval methods play a more crucial role in identifying the most relevant passages, and subsequently design a retrieval approach that weights different retrieval methods proportionally to their actual importance.

The goal is to develop a hybrid approach (our Neural Network) that does not simply assign equal weight to each retrieval method but instead combines their relevance signals in a way that reflects their actual importance. By learning how different retrieval techniques contribute to finding the best answer, this model can adjust their influence, leading to a more precise and efficient retrieval process. A detailed overview of the retrieval pipeline is provided in 1.4.

1.1.3 Comparison with traditional RAG paradigms

The final objective is to understand whether structuring and enriching documents can offer a valuable benefit in RAG systems.

To do this, we compare the performance of our proposed system against a “naive” RAG system, where the KB consists solely of textual chunks represented in vector form. The evaluation is conducted in two parts. The first part assesses the pipeline’s performance on punctual questions, i.e. questions whose answers are explicitly stated in the document and that do not require additional reasoning. We use question-answer pairs coming from the development set of Google’s Natural Question (NQ)[22] dataset, with the corresponding Wikipedia source pages constituting our initial unstructured encyclopedic collection. The second part of the evaluation focuses instead on Multi-Hop questions, which require retrieving multiple passages from one or more documents and performing an additional reasoning step to derive the answer. We generate Multi-Hop question-answer pairs synthetically from our KB. The quality of answers from both our pipeline and the naive RAG system is compared against the ground truth using three metrics: ROUGE, cosine similarity, and a judgment by an LLM with RAGAS[23], an automatic RAG evaluation system.

1.2 Key Research Questions and Thesis Outline

Here below we summarize the three **main research questions** of our experiment:

1. **Document Structuring:** How can unstructured encyclopedic documents be artificially structured and organized in a KB capable of making explicit all of the document’s informational dimensions (content, structure and entities)?
2. **Hybrid Retrieval:** How can a retrieval component be designed to take full advantage not only of the content of a document, but also of these enriching information?
 - (a) Which retrieval methods play a more crucial role in identifying the most relevant passages? How can a retrieval component optimally weight different retrieval methods proportionally to their actual importance in identifying them?

3. **Comparison with traditional RAG:** Overall, does the enrichment of a collection of document in this manner lay the groundwork for a retrieval system that can actually provide a significant benefit in terms of the retrieval of a correct passage and the quality of the system’s answers?

In chapter 2, we will provide an in-depth overview of the state of the art on the key topics covered in this thesis. We will begin by introducing the QA task, LLM, and their connections to KGs. Following this, we will examine how the central research questions of our study have been approached in previous work. Particular attention will be given to the limitations of existing methods, allowing us to identify open challenges that remain unresolved and where further contributions can be made.

In Section 3, we will outline the methodology adopted to tackle our research questions and objectives, detailing how we structure our KG and approach retrieval to effectively integrate its various aspects. Section 4 will focus on the practical implementation of the experiment, analyzing the results and validating the effectiveness of our proposed solutions. Finally, in chapter 5, we will present our conclusions, discussing the limitations of our approach and identifying potential directions for future research.

Chapter 2

Related Work on Question Answering Systems

This section provides an overview of the research landscape relevant to the development of our project. We first examine the broader context of Question Answering systems in section 2.1, tracing their evolution from early rule-based approaches to the era of machine learning. We will come in this way to talk, in section 2.2, about LLMs, how they work, and how they revolutionized the task of QA, placing particular emphasis on how they acquire and store knowledge. This in-depth comprehension of how these models work will help the readers understand why they face issues like hallucinations and outdated information. To address these issues, the integration of retrieval mechanisms with LLMs has given rise to Hybrid QA systems, presented in section 2.3, which combine the reasoning abilities of language models with the dynamic and precise Information Retrieval (IR) of external databases. This union between LLMs and retrieval systems will serve as a conceptual bridge to the introduction, in section 2.4, of RAG systems: a framework that harnesses retrieval systems to enrich the generative capacities of LLMs. Lastly, in section 2.5 we will delve into Knowledge Graphs, which are one of the most widely recognized methods for structuring information. Their significance lies not only in their broad utilization but also in their practical application within our own approach, where they serve as a foundational element for organizing and connecting data. In this way, we aim to provide readers with an overview of the main fields relevant to our experiment. Of course, experienced readers already familiar with these topics may choose to skip this preamble.

Building on this foundation, in section 2.6 we turn our focus to the first question of our experiment, exploring various methods for structuring and organizing unstructured documents. This section will not only serve as a

source of inspiration but also as a critical analysis to identify the limitations of these approaches, paving the way for improvements in our implementation.

Through this exploration, we will naturally transition to discussing QA systems specifically designed around graph databases and Graph-RAG systems. By combining the retrieval efficiency of graph-based structures with the generative power of LLMs, Graph-RAG systems offer a promising approach for addressing the limitations of traditional QA paradigms. The literature will provide the conceptual framework for our second research question, as we will examine how different authors handle retrieval in their databases, with particular attention to hybrid systems that combine knowledge from multiple sources. This analysis, described in section 2.7, will help us understand the strengths and weaknesses of existing approaches, informing the development of our own Graph-RAG system. Finally, in section 2.8, we will summarize the key findings on document structuring and hybrid retrieval, highlighting the key open challenges.

2.1 Overview of QA systems

Question Answering is a subfield of Natural Language Processing (NLP) and IR designed to generate precise answers to users' queries expressed in natural language. IR systems, like traditional search engines, are in fact not designed to return an answer, but to retrieve a list of documents ranked on their relevancy relative to a user's query. QA systems, instead, aim to provide precise responses, often in the form of a sentence or a paragraph, being in this way more aligned to users' needs and expectations[24].

Before the advent of LLMs, QA systems were composed of three distinct modules: question classification, in which the query is classified in a specific set of types, an IR system, used to retrieve the most relevant documents, and an answer extraction systems, that extracts the answer from the retrieved documents based on the query type[25]. To perform this last step, previous QA systems relied on NER, Part-of-Speech (POS) taggers and simple keyword matches to identify and extract only the relevant word or phrase. For example, systems like AskMSR[26] demonstrated as early as 2002 that a lightweight, pattern-based approach could deliver competitive performance by leveraging redundancy - i.e. identifying repeated occurrences of similar answers across multiple snippets - without relying on deep linguistic processing.

The introduction of machine learning methods, particularly deep learning, marked a paradigm shift. LLMs, particularly those based on the transformer architecture, quickly revolutionized the field of NLP and QA with

it. Models like Bidirectional Encoder Representations from Transformers (BERT)[27] demonstrated unprecedented performance in QA benchmarks such as SQuAD (Stanford Question Answering Dataset)[28], setting new standards in understanding context and semantics.

2.2 Large Language Models for QA

On a high-level overview, LLMs are probabilistic models that are trained to predict the most probable word (more properly a “token”) conditioned on a previous sequence of words.

LLMs are the result of decades of progress in neural network architectures, but the real turning point came with the introduction of the transformer architecture, detailed in the paper “Attention is All You Need” by Vaswani et al. in 2017 [29]. The transformer architecture quickly became the cornerstone of artificial intelligence, revolutionizing the field with its versatility and performance. Not only it was incredibly efficient for NLP tasks [30], but its adoption extended far beyond, finding applications across virtually every domain of AI, including computer vision, speech recognition and reinforcement learning. While an in-depth explanation of how LLMs are trained and how they work is out of the scope of this thesis, we are going to briefly introduce the transformer architecture in order to explain how they are able to store and recall facts and subsequently answer the user’s question. By comprehending how knowledge is stored inside these models, we believe that the reader will gain a deeper understanding of the beauty as well as the limitations of parametric memory and the consequent necessity for complementary systems like RAGs.

The protagonist of this revolution is attention, a mechanism that allowed models to focus on different parts of the input sequence simultaneously, rather than sequentially. The self-attention mechanism enabled transformers to efficiently capture long-range dependencies by assigning varying levels of importance to different tokens within a sequence, regardless of their position. In particular, multi-head self-attention allows the model to learn different types of relationships in parallel. These features not only addressed the limitations of RNNs but also significantly improved the computational efficiency and scalability of language models.

Along with attentions layers, transformers use Feed Forward Layers to extract additional patterns from the training data. The feed-forward network typically consists of two linear layers separated by a non-linear activation function, commonly ReLU (Rectified Linear Unit). Recent findings, like the study from Meng et. al. [31], suggest that in transformers facts are primarily

stored in the middle layers of these Feed Forward Networks rather than in the attention mechanism.

Given their behavior, from the inception of transformer-based LLMs the primary thrust was on assimilating additional knowledge through Pre-Training, essentially maximizing the networks’ capacity of recalling facts by augmenting its parameter and the data used for training. Many papers delved into exploring the advantages and limits of this approach. Petroni et. al. [32] present an in-depth analysis of the relational knowledge already present (without fine-tuning) in a wide range of state-of-the-art pretrained language models. They find that without fine-tuning, BERT contains relational knowledge competitive with traditional NLP methods that have some access to oracle knowledge. Moreover, they find that BERT also does remarkably well on open-domain QA against a supervised baseline. Roberts et. al.[33] fine-tune pretrained models to answer questions without access to any external context or knowledge. They show that this approach scales with model size and performs competitively with open-domain systems that explicitly retrieve answers from an external knowledge source when answering questions. That is not to say, though, that LLMs can solve the task of QA by themselves. This last study, in fact, presents in its conclusions multiple concerning problems that their approach faces:

1. A model of that size can be prohibitively expensive.
2. While “open-book” models provide some indication of what information they accessed when answering a question, an LLM distributes knowledge in its parameters in an inexplicable way.
3. LLMs tend to produce “hallucinations”, i.e. realistic-looking but false answers, when they are unsure.
4. An approach of this type gives no guarantees as to whether or not a model will learn a fact. This prevents the users from explicitly updating or removing knowledge from a pre-trained model.

As one may assess, these are serious problems that make these models inconvenient for numerous tasks[34], and hence need to be addressed.

2.3 Hybrid QA Models

Numerous past studies have developed architectures that integrate non-parametric memories into systems trained from scratch specifically for certain tasks. These methods include memory networks[35][36], stack-augmented networks[37], and memory layers[38].

Hybrid Models, though, gained much more traction, thanks to their ability to address most of the issues that LLMs face. These models, in general, work by combining parametric and non-parametric memory, essentially leveraging a retrieval engine just like pre-LLM QA systems. Instead of extracting the exact word or phrase from the retrieved documents, though, they condition generative models on these contexts in order to generate (or extract) the answer. This approach allows knowledge to be not only directly revised and expanded but also accessed for inspection and interpretation[3].

In 2019, K. Lee, M. Chang and K. Toutanova introduced ORQA (Open-Retrieval QA), a novel approach for tackling open-domain QA in a weakly supervised setting[39]. In this approach, the retriever uses a dense representation model, encoding both the question and candidate documents into a shared embedding space with a dual-encoder architecture. Candidate documents are then retrieved based on their similarity using approximate nearest neighbors. ORQA is also one of the first methods that uses a transformer-based LLM (BERT) as the reader module, to process the retrieved documents and predict the most likely answer span. All of these architectural choices, combined with a jointly training of the retriever and reader components, allowed ORQA to achieve state-of-the-art results on several open-domain QA datasets, such as NQ[22] and TriviaQA. In this way, this paper popularized in the field the use of dense vector-based models for the retriever module and pre-trained language models for the reader module, paving the way for subsequent advancements.

Retrieval Augmented Language Model pretraining (REALM)[40], for example, builds upon ORQA insights by showing for the first time how to pre-train a knowledge retriever in an unsupervised manner, using masked language modeling as the learning signal and backpropagating through a retrieval step that considers millions of documents. Dense Passage Retrieval (DPR)[41], instead, focuses on further refining the retrieval and extraction pipelines. While in ORQA the retriever is optimized indirectly via weak supervision from the reader, the authors of DPR introduced explicit supervision for the retriever by using labeled positive passages and hard negatives. By training the retriever directly with contrastive learning, DPR achieves more effective and faster optimization compared to ORQA’s latent objective. DPR also employs FAISS(Facebook AI Similarity Search)[42] for an approximate

nearest neighbor (ANN) search, enabling efficient retrieval from large corpora with billions of passages. This makes DPR far more scalable and practical for real-world open-domain QA tasks compared to ORQA. This work consolidated dense retrieval as a core technique for open-domain QA, serving as one of the main inspiration for subsequent research, including the conception of Retrieval-Augmented Generation.

2.4 Retrieval Augmented Generation for QA

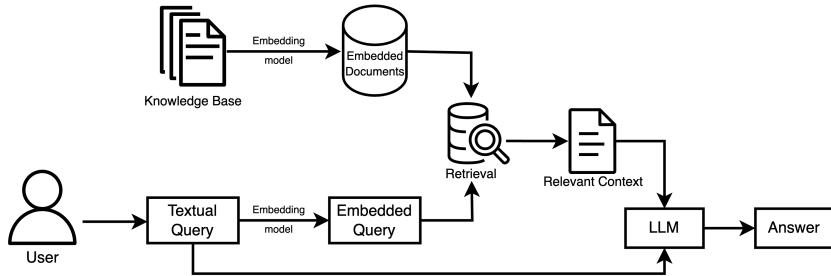


Figure 2.1: RAG architecture

Retrieval Augmented Generation was introduced in 2020 by Lewis et al. in the seminal paper “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”[3]. The main idea behind this work was to build on the concept of combining language models with a differentiable retriever, that REALM and ORQA proved to be promising, and extend it to the field of Seq2Seq modeling. To achieve that, the authors employed a retriever based on DPR and paired it with a Seq2Seq model, BART[43], to condition the generation of responses on both the input query and relevant retrieved documents. RAG enhances the capabilities of generative models by augmenting them with relevant, external information retrieved from a large knowledge base (e.g., document corpora or search engines). This integration enables the model to produce responses that are more accurate, diverse, and informative compared to standalone language models.

The pipeline, illustrated in figure 2.1, is often divided in literature into three phases. Prior to execution, the documents composing the KB are embedded using an embedding model. The pipeline then proceeds as follows:

1. In the **Retrieval Phase** the user’s query is embedded and used to retrieve the most relevant documents from the KB based on similarity in the embedding space. Often, the same embedding model is used

both on the documents and the query, to ensure that the generated vectors are aligned.

2. In the **Augmentation Phase** the retrieved documents are combined with the query and injected as additional context into the language model.
3. In the **Generation Phase** the language model generates a response to the user’s query conditioned on the retrieved context.

In their paper, Lewis et al. demonstrate that this simple paradigm can achieve state-of-the-art results on open NQ, WebQuestions and CuratedTrec and strongly outperform recent approaches that use specialized pre-training objectives on TriviaQA. Notably, RAG produces responses that are more factual, specific, and diverse than a BART baseline. Moreover, they demonstrate how the non-parametric memory can be replaced to update the models’ knowledge, essentially resolving one of the key issues of LLMs. As for what regards the other problems that LLMs face, just like hybrid QA systems, RAG drastically lowers the probability of producing hallucinations and is able to provide indication of what information it accessed when answering a question.

Since the effectiveness of this paradigm was evident, numerous research emerged in the following years to try and improve all the stages of the pipeline. Notably, the “Naive” RAG (the name that will be given to the initial architecture to differentiate it from subsequent improvements) suffered from challenges in all of its stages. Y. Gao et al. have compiled a comprehensive survey[4] regarding these issues and how they have been dealt in later advanced iteration of the RAG paradigm.

One way to enhance retrieval is by refining the KB itself. This can involve optimizing data indexing by adjusting chunk sizes, adding metadata (e.g., dates or chapter references), or improving the KB’s quality. Steps such as removing irrelevant information, verifying factual accuracy, and updating outdated documents have proven effective in boosting retrieval performance.

More advanced techniques focus on improving the embedding model or the retriever. Fine-tuning the embedding model, for example, can significantly enhance relevance in domain-specific contexts. In fact, while state-of-the-art models like ada-v2[44], AngIE[45], Voyage[46], and FlagEmbedding[47] are trained on extensive corpora, their ability to capture domain-specific nuances is limited. Fine-tuning these models on a specialized dataset aligns them with the target domain’s requirements.

Query rewriting is another effective approach that modifies user queries to better align them with relevant documents. For instance, Hypothetical

Document Embeddings (HyDE), introduced by Gao et al.[48], generate hypothetical responses to queries using an LLM. These responses are embedded and used to retrieve similar documents from the KB. The idea is that LLM-generated responses to the query could be closer in the vector space to the relevant documents than the query itself. While this has proven to be useful in certain cases, when the LLM lacks domain-specific knowledge it may generate inaccurate responses, which could lead to the retrieval of irrelevant documents.

For systems designed to answer precise and factual questions, replacing the retriever with a “Search Module” can be beneficial. Such modules use LLMs to generate structured queries in languages like SQL or Cypher[8]. We will return to this approach when talking about the retrieval component in RAG systems based on KGs.

In the augmentation and generation stage redundancy and repetition are concerns, as they could introduce noise and make the LLM loose focus on crucial information. Prompt Compression is a post-processing technique that involves compressing irrelevant context, highlighting pivotal paragraphs and reducing the overall context length. Approaches such as Selective Context[49] and LLMLingua [50] utilize small language models to calculate prompt mutual information or perplexity, estimating element importance.

Another set of techniques developed to tackle this issue are re-rankers, which proved to be a key strategy to improve RAG systems. Re-ranking consists in post-processing the retrieved documents in order to perform a second, more precise, screening. Diversity Ranker¹, for example, prioritizes reordering based on document diversity, in order to avoid information repetition. LostInTheMiddleRanker², instead, has been developed to alternate the placement of the best document at the beginning and end of the context window. This functioning is based on the notion that the order in which information is presented can directly influence the context passages on which the model places the most emphasis while generating its response. The study of N. Liu et al. [51] highlights in fact that language models often struggle to effectively utilize information placed in the middle of long contexts, suggesting that placing key information at the beginning or end of the context can lead to better performance.

¹<https://haystack.deepset.ai/blog/enhancing-rag-pipelines-in-haystack>

²<https://docs.haystack.deepset.ai/docs/lostinthemiddleranker>

2.5 Knowledge Graphs for QA

Knowledge Graphs are one of the most widely recognized methods for structuring information. The organization of data in graph form is so intuitive to humans that the concept of Knowledge Graphs is a well-established technology that has been in use for quite some time. However, it was the advent of the Semantic Web and linked data principles in the early 2000s that propelled the widespread adoption of KGs. Initiatives like Resource Description Framework (RDF)³ and Web Ontology Language (OWL)⁴ provided formal frameworks for defining and querying structured knowledge. The concept of “Knowledge Graph” gained even more traction among the public in more recent years. In particular, it was the announcement of “Google Knowledge Graph” [52] in 2012 and the following development of the same tool by major companies like Airbnb, Amazon, eBay, FaceBook, iBM, LinkedIn, Microsoft, that made the concept so popular[53].

KGs are a network of interconnected nodes and edges, where nodes represent entities (such as people, places, or concepts) and edges represent the relationships or connections between those entities. Formally, KGs store structured knowledge as a set of triples $KG = \{(h, r, t) \subseteq \varepsilon \times R \times \varepsilon\}$, where ε and R respectively denote the set of entities and relations. This particular way of representing information is what makes KGs useful, unique and, still today, a very powerful tool for representing and organizing complex information. KGs structure human concepts in a way that makes it possible for machines to “reason” over them: by formalizing relationships between entities, KGs allow algorithms to infer new insights, perform logical reasoning, and draw connections across disparate pieces of information. For these reasons, they find applications in a wide range of fields: they power search engines like Google, enable personalized recommendations in e-commerce, enhance data governance practices, and underpin fraud detection systems. In conversational AI, they provide context-aware responses, while in critical decision support systems, such as those used in healthcare, finance or supply chain management, they offer a reliable framework for synthesizing information.

S. Pan et al.[54] present a roadmap for unifying KGs and LLMs, providing insights of state-of-the-art systems that combine these tools and demonstrating future research directions. As the authors highlight, the potential of this combination lies in the fact that these two tools are perfectly complementary, with the weaknesses of one tool being the strengths of the other. KGs are a

³<https://www.w3.org/RDF/>

⁴<https://www.w3.org/OWL/>

rigid and structured way of representing knowledge, making them accurate and easy to interpret. On the other hand, LLMs have their knowledge stored implicitly in their parameters, making them prone to hallucination and impossible to interpret. LLMs can also lack domain specific or new knowledge, as they operate within the constraints of a knowledge cutoff, while KGs can be easily tailored to one specific domain and can effortlessly be updated. That is not to say that KGs are perfect. They can often be incomplete and, when this happens, they have no capacity of reasoning over lacking information. In this respect, since LLMs are trained on impressive amount of information, they are packed with general knowledge and are able to generalize to unseen facts, making them exactly complementary to KGs' weaknesses.

In their roadmap, the authors identify three frameworks in which research directions can be divided:

1. **KG-enhanced LLMs**

Includes research efforts that leverage the capabilities of KGs to address challenges associated with LLMs.

- (a) *KG-enhanced LLM pre-training* enhances the pre-training stage of the LLM with knowledge from KGs[55][56].
- (b) *KG-enhanced LLM inference* consists in retrieving knowledge from KGs and incorporating it into the LLM during inference, in this way enabling them to access knowledge without retraining. This is the main focus of this thesis, to which the next chapters are going to be dedicated.
- (c) *KG-enhanced LLM interpretability* uses KGs to understand the knowledge learned by LLMs[32] and their reasoning process[57].

2. **LLM-augmented KGs**

Includes research efforts that leverage the capabilities of LLMs to address challenges associated with KGs. These includes enriching KGs with embeddings[58], constructing KGs from unstructured natural language documents[59] and generating coherent natural language texts from existing KGs[60][61].

3. **Synergized LLMs + KGs**

Explores approaches that integrate LLMs and KGs to leverage the strengths of both paradigms, allowing each to address the limitations of the other. One prominent direction is *Synergized Knowledge Representation*, which aims to develop unified models capable of capturing and representing knowledge from both text corpora (implicit and unstructured) and KGs (explicit and structured)[55][62][63]. Another key focus

is *Synergized Reasoning* that focuses instead on design models that can effectively conduct reasoning with both LLMs and KGs[64][65].

Although this thesis focuses on KG-enhanced LLM inference, it is important to provide a brief overview of KG construction from unstructured text. While we will not directly implement these technologies, relying instead on Wikipedia’s hyperlinks to extract and link entities, readers aiming to replicate our findings with different documents may need to apply these techniques to structure information in a similar manner.

During the years, many public KGs designed to store as much knowledge as possible have been developed through collaborative effort and through the development of automatic extraction tools. Encyclopedic KGs, like DBpedia[66], Freebase[67] and WikiData[68], have been built to cover factual or event knowledge from different domains. Linguistic KGs, like WordNet[69], were instead built to store semantic relations between words, and are often used to create high-performance word embeddings. However, the vast majority of information still exists in the form of unstructured text. While RAG frameworks are becoming more and more popular in industries, the adoption of their combination with KG is undermined by the difficulties that arise in building the latter. For this reason, extracting structured knowledge from unstructured text has become a crucial task in NLP and has garnered significant attention from both academia and industry.

This field of academic literature is called Automatic Knowledge Graph Construction (AKGC), and is defined as the process of constructing KGs from unstructured text[18]. Traditional approaches to RDF heavily relied on rule-based methods and hand-crafted features, therefore requiring extensive domain expertise and being limited in their ability to scale to large datasets. With the advent of deep learning and the advancements in NLP, the task of RDF shifted into a multiple-step pipeline that first discovers (NER) and links (NER) conceptual entities, resolves coreference mentions (Coreference Resolution), and finally extracts relationships among entities (Relation Extraction)[18]. This demonstrates, though, how complex developing an AKGC tool can be, as it requires to build multiple models (one for each step) and join them together in a cohesive pipeline. Companies and research groups that may want to implement KGs in their solutions may be discouraged by the effort required to transform their documents into structured form.

In more recent years, though, researchers have validated the possibility to use LLMs to construct a KG from documents written in natural language. This is the case, for example, of the work of Kumar et al.[59], that uses one fine-tuned LLM for entity extraction and a second LLM for relation extraction, or Han et al., that proposes a prompting technique with an iterative

verification framework. This represents an important shift in the field, as it significantly lowers the entry barrier, thus allowing companies and researchers to build their KGs from documents without any excessive effort.

The abilities that LLMs showcase have brought the community to use them as well, in combination with clever prompt engineering, as KG constructors from texts⁵⁶. Although the results may not be as precise, this method permits to extract triples from text in a fast and efficient way, lowering the required effort even further. This simple approach gained so much traction that Neo4j itself implemented LangChain’s graph transformers⁷ to allow their users to build graphs directly from documents. The key takeaway here is that anyone wishing to develop a solution using a Knowledge Graph should no longer feel discouraged by the effort required in its construction, since tools such as those just mentioned are available to make the task as easy as it has ever been.

2.6 Document structuring for QA

Table 2.1 provides a summary of state-of-the-art methods for document structuring in QA. Our primary goal is to assess how each approach preserves and highlights the different informational dimensions of the original documents. As we will observe in this section, most methods focus on a single dimension, prioritizing what they consider most relevant for their specific use case. Only a few of them attempt to retain multiple aspects of the original documents, yet they often come with certain limitations.

As said in 2.4, the “naive” RAG approach involves segmenting the text into smaller chunks and converting these into vector representations stored in a vector database. The ease of implementation of this method has made it exceptionally common among businesses and researchers. This process typically focuses on the semantic embedding of each chunk, which determines its placement in the vector space. However, this method often results in chunks being loosely connected, with their organization relying mainly on the semantic proximity in the embedding space. Any additional structure or context from the original document is usually preserved only through basic metadata, such as document identifiers or segment annotations, leading to a loss of the broader document hierarchy and context.

⁵Rahul Nayak in “Towards Data Science”, Nov 10 2023, towardsdatascience.com/how-to-convert-any-text-into-a-graph-of-concepts

⁶LangChain LLMGraphTransformer documentation: langchain.com/en/latest/graph-transformers

⁷<https://github.com/neo4j-labs/llm-graph-builder>

Method	Approach	Description	Content	Structure and Organization	Entities and Semantic Relations	Other Limitations
Naive	DPR[41], RAG[3]	Splits text into chunks of similar dimensionality.	Maintained in chunks.	Can be maintained only by basic metadata or annotations.	Lost	-
Segmentation	ULSSoLD[70], Lumber Chunker[71]	Identify and classify semantic section of documents.	Maintained in chunks.	Chunks and their classification can reflect relationships and hierarchies.	Lost	While segmented and labeled, sections are not organized in a more explicit structure.
Traditional KG	AKGC[18]	Represent information in atomic formatting: Subj, Pred, Obj.	Not all information can be represented in atomic form.	Lost	Maintained in nodes and edges.	-
Dual Layer	KG-Retriever[72]	Bottom Layer: Traditional KG Upper Layer: document similarities.	Not all information can be represented in atomic form.	Maintained in the upper layer, but sections are lost.	Maintained in the bottom layer.	-
Hybrid	Knowled-GPT[8]	Traditional KG enhanced with some triples having the object as a chunk of text.	Maintained in enhanced nodes and relations.	Lost.	Maintained in the traditional node and edge structure.	KG structuring is not rigid, but determined by an LLM on a case-by-case basis.
	GraphRAG[73]	Traditional KG enhanced by enriching nodes and relationships with textual descriptions.	Maintained in enhanced nodes and relations.	Graph Communities are created, but the structure of the original doc is lost.	Maintained in the traditional node and edge structure.	-
Chunk Graph	DHR[74]	Structure documents as hierarchical trees with document titles as roots, section titles as intermediate nodes, and chunks as a leafs.	Maintained in nodes.	Maintained in the structure of the KG	Lost.	-
	customer KG-RAG[13]	Structured KG for customer service tickets.	Maintained in nodes.	Maintained in the structure of the KG	Entities are lost. Semantic relations are maintained only in the form of semantic similarities.	The structure is rigid and specific for tickets.
	KGP[75]	Documents are maintained and connected to their chunk nodes. Chunks are linked based on their lexical and semantic similarity.	Maintained in nodes.	Each document is connected to their respective chunk, but sections are lost.	Entities are extracted, but not stored, to connect lexically similar chunks. Semantic relations are maintained only in the form of semantic similarities.	Documents are not interconnected, but linked solely to their respective chunks.
	Docs2KG[76]	Identifies chapters, subchapters, paragraphs, and sentences as nodes. Links them with structural and semantic relations.	Maintained in nodes.	Maintained through "has-child", "before" and "after" relations.	Entities are not extracted. Semantic relations are limited to "same time", "focus", "support by", "explain".	-

Table 2.1: State-of-the-Art Methods for Document Structuring

For each approach, we describe whether and how it preserves the three informational dimensions of the original document: content, structure and entities.

That is not to say the the simple division of the content into chunks is in total contraposition to a more in depth understanding of the structure of the document. M Rahman and T. Finin[70], for example, develop a framework that can analyze a large document and help the user find where a particular information is located. They do this by employing Deep Learning models to automatically identify and classify semantic sections of documents while assigning consistent and human-understandable labels. Another example of a solution that tries to define a more organized structure while still focusing primary on the textual content is LumberChunker[71], a method that leverages an LLM to dynamically segment documents. LumberChunker works by iteratively prompting the LLM to pinpoint the exact location within a series of sequential passages where a shift in content begins. These methods go beyond merely splitting the text into chunks: they introduce sophisticated techniques that aim to better preserve the logical and semantic structure of the original document. By leveraging these techniques, the resulting chunks are not just isolated pieces of text but are carefully segmented to reflect deeper relationships and hierarchies within the document, enhancing the overall coherence and retrieval effectiveness. Though, they only focus on segmenting and labeling passages of a document, without organizing them in a more explicit structure.

As we discussed in section 2.5, KGs are instead a significantly more structured way of representing information. AKGC tools[18] typically focus on constructing traditional KGs, where the information is represented as Subject-Predicate-Object triples, with both Subject and Object being atomic entities. This approach involves extracting entities and relationships from the original text., with the goal of transforming the document’s content into a structured representation of facts through these triples. While this traditional structure proved to be efficient multiple times, when extracting triples from a set of documents we completely loose the text’s organization.

An attempt to solve this issue, while retaining the traditional graph structure composed of atomic entities, is presented in KG-Retriever[72]. In it, authors W. Chen et al. propose a dual layer structure. In the bottom layer, relationships between entities mentioned within each document are modeled by an LLM inside an entity-level graph. In the upper layer, a document-level graph is constructed to improve RAG’s capability of integrating information across multiple documents, with each edge representing the similarity between the corresponding records. Each document in the upper layer is then connected to the corresponding KG in the bottom layer.

Though, the authors of KnowledGPT[8] argue that a significant portion of information can hardly be represented in this “atomic” formatting anyway. For this reason, they enrich their traditional KG by proposing an additional

knowledge representation, termed as “entity-aspect information”, that consists in a variation of triple where the object is a long piece of text. For example, the node “Harry Potter” could be connected via a “description” relation to an entire paragraph from the book describing his appearance (e.g., “Harry had always been small and skinny for his age...”), rather than being constrained to extract atomic entities and relationships (e.g., “Harry Potter”, body_type, “skinny”), which the authors argue that might risk losing important contextual or implicit information.

While it is not the central focus of the paper, they prove their argument by investigating the knowledge extraction coverage of their tool on 100 documents from HotpotQA[77] by employing the word recall rate, computed as

$$\frac{|W_{extracted} \cap W_{doc}|}{W_{doc}}$$

where $W_{extracted}$ and W_{doc} denote the set of words in the extracted knowledge and the starting document respectively. Preprocessing was applied by removing stop word and applying lemmatization. Results show that the extraction coverage stands at 0.53 when solely relying on triples representation. This indicates that only a limited portion of knowledge can be represented in such form. With the employment of entity-aspect information and entity descriptions this metrics rise by about thirty percentage points, suggesting that incorporating textual nodes enables KnowledGPT to populate the KB with a broader spectrum of knowledge.

What remains unexplored in the KnowledGPT paper is how textual chunk nodes can be systematically integrated into a well-defined and organized structure. The process of knowledge extraction from documents is in fact delegated to an LLM, which is provided with various possible representations of the document, such as entity descriptions, relational triples, or entity aspect information. As a result, this representation is not systematically structured but is instead determined by the LLM on a case-by-case basis. Moreover, while a chunk node may be linked to an entity via its predicate, it loses any contextual information regarding its original position within the document, leading to a disconnection from the document’s inherent structure. These shortcomings are also shared by approaches like Graph RAG by D. Edge et al.[73] which, despite attempting to enhance their representation by enriching both entities and relationships with descriptions, fail to preserve the data concerning how that information was organized in the source document.

While both KnowledGPT and Graph RAG strike a balance between traditional entity nodes and chunk nodes, other approaches prioritize chunk nodes as the core of their representation. For instance, DHR[74] prioritizes preserving the hierarchical structure of the original document at the expense

of entity information and their semantic relationships. In this approach, documents are modeled as hierarchical trees, where document titles serve as root nodes, section titles as intermediate nodes, and chunks as leaf nodes.

In contrast, Z. Xu et al. [13] propose a representation that does not strictly focus on a single informational dimension of the original document but instead seeks to preserve aspects of each. Their work focuses on transforming unstructured ticket data into a well-defined KB, enhancing the information with additional context and relationships. Similar to the approach taken by KG-retriever, the authors focus on capturing both intra-document and inter-document relationships, but they integrate these within a unified KG for a more cohesive representation. For intra-document relationships a hybrid approach is used, starting with rule-based extraction for predefined fields that can be identified by keywords and continuing with the employment of an LLM for more complex unstructured text. The LLM is guided by a YAML template that maps common ticket sections into graph structures, ensuring consistency in how ticket information is organized. For the parsing of inter-ticket connections, explicit relations are established based on predefined fields, such as those found in systems like Jira. Additionally, implicit connections are inferred by analyzing textual and semantic similarities between ticket titles, employing embedding techniques and a threshold mechanism. The resulting structure is well organized, but its fields are rigid and specific for tickets, making this approach hardly generalizable to other types of documents.

A focus on the generalizability of the approach to more generic initial structures is provided by Y. Wang et al. in their paper KGP[75], which defines a pipeline that, like ours, begins instead with generic unstructured documents. Similar to our approach, their KB is structured as a graph where nodes represent records, text passages, or tables, albeit without subchapter divisions. However, their organization differs significantly. In KGP, documents are not interconnected, but linked solely to their respective chunks and tables. What binds the whole graph together are passage nodes, which are linked together, even if belonging to different documents, via edges denoting their lexical or semantic similarity. For lexical similarity, they utilize Term Frequency-Inverse Document Frequency (TF-IDF) and TAGME[78] to extract keywords and Wikipedia entities, connecting two passages if they share any common element. For semantic similarity, they embed chunks and connect those with close vectors. Our approach could improve upon this method by preserving the document structure more effectively through the use of subchapter nodes. Additionally, while KGP extracts Wikipedia entities, it does not incorporate them as nodes. Not only we do that, but our approach, though less generalizable due to its focus on encyclopedic docu-

ments, benefits from the overlap between the concepts of pages and entities. This allows us to build a more cohesive graph, as chunks from one document can directly cite a second document, and documents themselves can be directly connected through DBpedia relationships.

The final approach presented, Docs2KG[76], shares our goal of building a structure capable of organizing unstructured documents, which represent the vast majority of business data. However, Docs2KG places more emphasis on extracting multimodal information from diverse and heterogeneous unstructured documents, such as emails, web pages, PDF files, and Excel spreadsheets. This focus contrasts with methods previously cited like that of Z. Xu, which primarily concentrates on a single source of information (tickets). For purely textual documents, which is the focus of the thesis, Docs2KG constructs the KG from two perspectives: one related to structure and content and the other to semantics. To maintain the document’s structure, Docs2KG identifies chapters, subchapters, paragraphs, and sentences as nodes, and links them with relations such as “has-child”, “before”, and “after”. On the other hand, in order to preserve the semantic aspect, the method extracts entities and connects them using relationships like “same time”, “focus”, “supported by”, and “explain”. This subdivision of information in structure, content and semantics is very close to our methodology. Though, the limited set of semantic relations is a significant drawback of Docs2KG, as it fails to fully explore the document’s deeper semantic connections, potentially limiting its understanding of the more complex relationships between the elements within the document.

2.7 Improving Retrieval in KGQA

Retrieval Type	Name	Approach Description	Focus	Limitation
Dense Passage Retrieval	Naive RAG[3]	Retrieval of chunks whose embedding is nearest to the query's embedding.	Content	Incomplete Doc Structuring (2.1)
	DHR[74]	A document-level retriever first identifies relevant documents, among which relevant passages are then retrieved by a passage-level retriever.	Structure	Incomplete Doc Structuring (2.1)
Search Module	CoKG[79]	Traditional Search Module: leverages the LLM to formulate structured queries in the appropriate language.	Entities	Applied to traditional KGs: incomplete doc structuring
	KnowledGPT[8]	PoT Prompting: generates search language for KBs in code format with pre-defined functions for KB operations.	Content and entities	Incomplete Doc Structuring (2.1)
SubGraph Retrieval	VRN-QA[80], SR[81], TransferNET[82]	First step: identification of the topic entities. Second step: SubGraph expansion for relevant nodes inclusion.	Entities	Applied to traditional KGs: incomplete doc structuring
	G-Retriever[9]	Price Collecting Steiner Tree (PCST) problem to directly extract a subgraph that encompasses as many relevant nodes and edges as possible.	Entities	Applied to traditional KGs: incomplete doc structuring
Hybrid Retrieval	HybridRag[83]	Vector Retrieval + Search Module (concatenation).	Content and entities	Same weights to all retrieval methods
	Doan et al.[84]	Embedding concatenation.	Content and entities	Same weights to all retrieval methods
	KGP[75], Docs2KG[76], customer KG-RAG[13]	Node Retrieval + Search Module filtering.	Content and structure	Incomplete Doc Structuring (2.1)

Table 2.2: Comparison of State-of-the-Art Methods for RAG retrieval

Except for Naive RAG, all the retrieval methods presented pertain to QA on KGs.

In this section, we review the literature on Knowledge Graph Question Answering (KGQA) and KG-based RAG to draw inspiration for the design of our KG-based retrieval algorithm.

KGQA is a subfield of QA that focuses on the development of systems that use a Knowledge Graph as their underlying source of knowledge. Al-

though some research in this area explores Semantic Parsing techniques, our focus will be on methods grounded in IR. Regarding instead KG-based RAG systems, the work of Y. Gao et al. [4], previously cited in this thesis, provides a broad analysis of the methodologies and includes a dedicated discussion on the integration of structured data. Additionally, the survey by T. Procko et al. [85] offers a comprehensive review specifically focused on graph-based techniques for RAG systems. Both surveys served as key references for identifying and contextualizing the papers discussed in this section.

The informational dimension that a retrieval system prioritizes is inherently determined by its underlying structure. As we observed in Section 2.6, the state of the art lacks a representation capable of incorporating all three informational dimensions, which means that no existing retrieval method can fully exploit them. However, we can take inspiration from state-of-the-art approaches to understand how each informational dimension is retrieved individually and use these insights as the foundation for our hybrid retrieval system, which will integrate them. Additionally, we can study existing hybrid retrieval systems to explore effective strategies for combining different retrieval methods.

2.7.1 Node Retrieval

The most straightforward approach to retrieval in a graph database, much like in vector databases, relies on embeddings. Specifically, the content of all graph nodes can be embedded beforehand to identify, at each iteration, the node most similar to the query based on embedding similarity. Once the most relevant node is found, a common approach is that of retrieving its neighborhood to provide contextual information. While this method is relatively simple, it first ensures that a pertinent entity is selected in response to the query and then provides back to the LLM all its relevant surrounding context.

DHR [74], cited in Section 2.6, proposed for document structuring a hierarchical tree structure where document titles serve as root nodes, section titles as intermediate nodes, and chunks as leaf nodes. For retrieval, this approach employs a two-stage retrieval process: first, a document-level retriever identifies relevant documents, then a passage-level retriever selects the most relevant child passages, using embedding-based similarity at both stages. In this way, the retriever effectively leverages the graph structure by first identifying a relevant node, extracting its neighborhood, and then further refining the selection of nodes within this neighborhood using the second passage-level retriever.

Of course, there are more complex ways to pinpoint the initial nodes.

A slightly more advanced approach leverages NER to extract the central entity of the user’s question and NER to link it to the corresponding node in the KG. As in the previous method, the retrieved node’s neighborhood then forms the context provided to the LLM for generation.

One of the studies referenced in Section 2.6, KGP [75], explores the effectiveness of such an approach. After constructing their hybrid KB, the authors employ a TF-IDF method to identify “seeding nodes”, i.e. the most relevant nodes for answering the user’s query. Their findings reveal that, for most questions, the supporting facts are fully covered within the neighbors of these seeding nodes. However, low precision indicates that many neighboring passages are irrelevant to the query. Consequently, retrieving all neighboring nodes indiscriminately introduces redundant information, ultimately hindering the LLM’s ability to generate accurate responses. To mitigate this issue, the authors propose finetuning an LLM to dynamically guide traversal over the graph, selecting only the most relevant passages based on the query and consequently reducing context size by preserving only the most informative nodes.

We leverage the insights from this study to refine our retrieval method within our RAG pipeline. By integrating different node retrieval strategies and relevance-based filtering techniques, we aim to enhance the selection process, ensuring that only the most contextually relevant nodes are retained. This approach optimizes the quality of retrieved information, ultimately improving the accuracy and efficiency of our system.

2.7.2 Search Module

A second widely adopted approach involves using an LLM to translate user queries formulated in natural language into structured queries expressed in a graph query language such as Cypher. This corresponds to what we previously introduced in Section 2.4 as the “Search Module”. The simplicity of this method, since its basic implementation requires only a single LLM completion, has contributed to its widespread adoption in the community, as seen in tools like LangChain and Neo4j.

An example of this approach in the literature is Chain-of-Knowledge [79], an iterative retrieval framework proposed by X. Li et al. The method begins by employing an LLM to generate preliminary rationales in response to a user query, helping to identify relevant knowledge domains. In the initial step, an Adaptive Query Generator leverages the LLM to formulate structured queries in the appropriate language, allowing the system to interact with variously structured knowledge sources. The retrieved information is then used to refine both the initial and subsequent rationales, mitigating error propagation

throughout the reasoning process. This iterative refinement continues until the reasoning chain converges on a final, well-supported answer.

In spite of the popularity of this approach, while trying to implement it in our pipeline it quickly became apparent that it wasn’t well-suited for the application on a database with our specific structure. While effective for traditional entity-based graphs with well-defined structures, this method struggles with a graph enriched by textual nodes and thousands of diverse relationships. The generated queries often proved inaccurate, as the LLM frequently hallucinated about traversable relationships, confusing them with existing ones unrelated to the starting node. Additionally, it faced challenges in comprehending the less rigid graph structure, such as subchapters with numerous unique titles. It’s important to clarify that this doesn’t imply that the method is unsuitable for Graph-RAG systems in general. In fact, when it works, it’s the only approach capable of answering questions about the graph’s structure, such as “How many pages are cited in this text chunk?” or “How many chapters does this Wikipedia page have?”. However, this method is inappropriate for the specific graph structure we are working with, and its primary advantage is less relevant given the type of questions in our dataset.

Also KnowledGPT[8], which we cited in section 2.6 as a key inspiration for structuring the KB, is an example of the translation of the user’s query into applicable retrieval actions through an LLM. Though, its approach surpasses the straightforward translation of the user query into Cypher queries, to accommodate for the more complex organization. In particular, the authors implement the Program-of-Thought prompting technique[86], which generates search language for KBs in code format with pre-defined functions for KB operations. The first of these predefined functions is used to link the entities mentioned in the query with the nodes of the KB. It works by first searching in the KB for candidate entities, then gathering information about all of the candidates (their entity description and triples information), and finally using the LLM to determine the most appropriate entity. A second function, called by the authors “`find_entity_or_value`”, is designed to retrieve the corresponding entity or attribute value based on a query composed of an entity and a relation. A third function, called “`find_relation`”, is instead designed to retrieve the corresponding relation based on a set of two entities. Leveraging these different possible operations, the model is able to traverse the graph, moving in disparate ways according to the type of question. Inspired by this work, we too have adapted the approach to better suit our specific needs. Rather than instructing the LLM to generate Cypher queries, we instead leverage it as a selector within a predefined set of candidate subchapters. This modification effectively mitigates the previously mentioned issues, ensuring more reliable retrieval and avoiding hallucinations while maintaining

the benefits of the introduction of a complex language model.

2.7.3 Subgraph Retrieval

A third widely adopted approach involves retrieving a subset of the original KG, i.e., a subgraph, that is most relevant to the given question. This method is highly flexible, encompassing techniques that range from simply extracting a subgraph composed of nodes and edges whose embeddings are most similar to the query to more advanced strategies that leverage Graph Neural Networks (GNNs).

It is worth noting that this approach is not entirely distinct from others, as Node Retrieval is often used to identify the starting points of subgraph expansion, commonly referred to as seed nodes or topic entities [80][9]. From these initial nodes, the subgraph can be expanded in a targeted manner. However, in this section, we focus specifically on research that constructs subgraphs primarily using embeddings. This choice aligns with our broader methodology, which prioritizes the simplest possible implementation of each retrieval technique, since the overall complexity of our system should emerge from the intelligent combination of multiple retrieval methods rather than from the sophistication of any single approach.

Zhang et al.[80], for example, introduce a Variational Reasoning Network (VRN) that is divided into two modules: one for the identification of the topic entities and one for logic reasoning over the KG. This latter component, starts from the identified topic entity and performs a topological sort for all entities within T hops, where T is assumed to be known by the algorithm. In this way it identifies a subgraph G_y which contains a set of candidate answers and uses a learned vector representation to score each path of G_y based on their compatibility with the question, coming in this way to the final answer. Similarly, Zhang et al.[81] propose a model that, starting from the seed nodes, retrieves connected entities and relations from the KB by expanding the subgraphs to include potentially relevant nodes within a certain hop distance. This is done by leveraging the similarity of the relations' and the question's embeddings.

Also TransferNET[82] computes a score for each relation to denote their activated probabilities in terms of the current query, but then uses them to transfer the entity scores across those activated relations. One notable thing about TransferNET is that the model at each step attends to a different part of the question to get the query vector. Repeating this process over several iterations makes it possible to essentially move across relations, ultimately leading to the target entity.

G-Retriever[9], instead, is the only cited approach that does not address

the problem in an iterative way, but instead tries to directly extract the most relevant subgraph as a whole. Authors He et al. aim at constructing a subgraph that encompasses as many relevant nodes and edges as possible, while keeping the graph to a manageable size. They do so by formulating it as a PCST problem[87] that balances two objectives: maximizing the prize values associated with its nodes and minimizing the cost of its edges, where a higher prize, in this case, is assigned to those nodes and edges that are more relevant to the query. More specifically, the most significant k nodes and edges are awarded prizes on a descending scale from k to 1, while all others receive a prize value of zero.

For our subgraph retrieval approach, we draw inspiration from both strategies. After identifying the initial seed nodes, we iteratively construct a subgraph based on embeddings, following the methodology outlined in the first three papers discussed in this section. However, we then introduce an additional selection step: once a subgraph has been built for each seed node, we compute an overall relevance measure (also embedding-based) to determine which subgraph is the most relevant as a whole.

2.7.4 Hybrid Retrieval

Not much research has been done on RAG systems that can combine effectively relevance information coming from different retrieval techniques.

One of the few studies addressing this challenge is HybridRag [83], which explores the combination of Vector-RAG and a Graph-RAG system which employs a Search Module. However, their approach only explores the effectiveness of simple context concatenation, merging the relevant information retrieved from both methods. Their findings demonstrate that this hybrid approach outperforms both Vector-RAG and Graph-RAG individually in both retrieval accuracy and answer generation quality.

A similar strategy is employed by tools like LangChain⁸, a general-purpose framework for building LLM applications. In one of their blog posts⁹, they describe how their framework facilitates KG-based RAG construction and propose a hybrid retrieval method that concatenates the contexts retrieved from both Vector-RAG and Graph-RAG. Unlike Hybrid-RAG, however, their retrieval process incorporates additional mechanisms: unstructured data is retrieved not only via embedding similarity but also using keyword-based techniques, while structured data retrieval relies on Node Re-

⁸langchain.com

⁹blog.langchain.dev/enhancing-rag-based-applications-accuracy-by-constructing-and-leveraging-knowledge-graphs/

trieval rather than a Search Module, specifically its variant that employs NER.

These studies highlight the potential of hybrid retrieval strategies but also expose a gap in research regarding more sophisticated techniques for integrating multiple relevance signals beyond simple context concatenation. Two key limitations emerge. First, these approaches typically consider only two sources of retrieval. Introducing additional sources would significantly increase the size of the context passed to the LLM, which could quickly become impractically large, potentially degrading the model’s ability to generate accurate responses. Second, these methods implicitly assume that the relevance signals from different sources contribute equally to determining the best overall answer, an assumption that may not always hold.

A more sophisticated approach is proposed by Doan et al. [84], who develop a method that integrates standard text embeddings with KG embeddings. Their process begins by retrieving the top-N passages using standard embeddings. A KG is then constructed from these passages and embedded separately. The text embeddings and their corresponding KG embeddings are subsequently concatenated to form a hybrid embedding. The query’s embedding is generated using the same procedure, and the final set of retrieved passages is determined based on their similarity to this hybrid query embedding. This approach represents an improvement over previous methods, since the final context is not merely a concatenation of independently retrieved chunks but rather a selection of the most relevant ones based on both retrieval strategies. Though, it still does not fully resolve the existing challenges. The inclusion of additional sources would still require increasingly long embeddings, making the method impractical at scale. Furthermore, the hybrid embeddings are constructed by combining the two sources in equal proportions, meaning that each source retains the same level of influence over the final selection of chunks, regardless of its actual relevance to the specific query.

More compelling examples of hybrid retrieval can be found in the literature on Document Structuring, which we previously introduced. This connection is unsurprising, as these studies work with KBs similar to ours, requiring a retrieval strategy akin to the one we aim to develop.

One such example is KGP[75], which we already cited in the section on Node Retrieval. By leveraging LLM-generated queries to filter a node’s neighborhood, KGP effectively implements a hybrid retrieval strategy, as it combines two distinct methods: node retrieval and search module. A similar approach is demonstrated by Docs2KG [76]. In it, Node Retrieval is applied to obtain an initial set of relevant nodes. These selected nodes then serve as anchors for a subsequent subgraph retrieval process, which expands the

selection through Multi-Hop queries generated by an LLM.

An even more refined approach is presented in the ticketing system study by Xu et al[13]. Instead of relying solely on embeddings, their method begins with an LLM extracting the intent and mapped entities from the user’s query. These extracted elements are then embedded and used for retrieval, identifying the most relevant ticket based on cosine similarity. The original query is then rephrased using an LLM, replacing the subject with the retrieved ticket ID. Finally, the reformulated question is transformed into a Cypher query, allowing the system to traverse the graph and reach the answer.

These studies exemplify effective hybrid retrieval, demonstrating how multiple relevance signals can be integrated without merely concatenating retrieved contexts or imposing a fixed balance between sources. However, they achieve this by constructing a retrieval pipeline in which each step sequentially filters the chunks passed from the previous stage. None of these approaches investigate the effectiveness of building a comprehensive relevance measure that directly integrates the contributions of multiple retrieval methods into a unified score.

2.8 Summary of Findings and Key Open Challenges

After introducing the fundamental tasks of QA and RAG, along with the necessary tools for their implementation, Section 2.6 explored state-of-the-art methods for representing and organizing unstructured documents, with a particular focus on how each approach preserves specific informational dimensions.

We discussed KGs as a primary resource for organizing data within a structured framework. However, structuring information within a graph is not always effective. This is especially true when dealing with traditional graphs that represent information using Subject-Predicate-Object triples, where both Subject and Object are atomic entities. The authors of KnowledgeGPT [8] demonstrate that a significant portion of information cannot be effectively represented in such an atomic format. This observation motivated our exploration of alternative strategies for structuring unstructured documents, with a particular focus on preserving and explicitly encoding information about document organization and the relationships between referenced entities.

Most of the studies addressing this problem opt for a hybrid KG approach, where nodes are not strictly atomic entities but instead contain entire tex-

tual chunks. While this approach helps retain more context, many existing methods suffer from notable limitations. For instance, approaches such as KnowledGPT itself fail to preserve structural information about the document’s layout, while others, like DHR, neglect entities and their relationships. Additionally, some other methods, like that developed by Xu et al., rely on highly specific heuristics, limiting their generalizability to diverse document types. Among the surveyed works, Docs2KG is the approach most similar to ours, as it constructs a KG from two perspectives: one capturing document structure and content and the other focusing on semantic relationships. However, Docs2KG employs a limited set of semantic relations, which we identify as a significant drawback.

If the KB does not retain all relevant information, the retrieval process cannot utilize it effectively. Therefore, the absence of a representation in the state of the art that preserves all informational dimensions of the original document directly translates into a lack of retrieval methods capable of leveraging them. In Section 2.7, we examined how each retrieval approach prioritizes one or a few informational dimensions, a choice inherently dictated by the underlying structure of the system. Nevertheless, we drew inspiration from existing works to introduce three main retrieval strategies, which we aim to integrate into our hybrid retrieval system:

1. **Node Retrieval** involves selecting an initial set of nodes based on embedding similarity or NER and NEL, followed by extracting their entire neighborhood. The goal is to retrieve a set of relevant entities and pass them to the LLM along with their surrounding context to enrich the understanding of those entities. KGP[75] demonstrates that, for most queries, the necessary supporting facts are present within the neighborhoods of the relevant nodes. However, it also highlights a low precision issue, as many of these neighboring passages turn out to be irrelevant to the query. These insights support our approach of inserting this technique into a hybrid approach. In fact, while node retrieval is effective in capturing relevant passages, combining it with additional methods allows for a more selective process, ensuring that only the most contextually relevant chunks are retained.
2. The **Search Module** involves leveraging an LLM to translate user queries into structured queries expressed in a graph query language such as Cypher or, more broadly, into applicable retrieval actions. This broader variant has been a major source of inspiration for us in designing a method that addresses the limitations of the first, which in our case produced unsatisfactory results.

3. The **Subgraph Retrieval** technique involves retrieving a subset of the original KG, i.e. a subgraph, that is most relevant to the question. From the literature, we identified two main approaches: iterative and non-iterative. Similarly to Node Retrieval, the iterative approach begins by identifying initial nodes, known as seeding nodes. From there, the subgraph is expanded iteratively, adding the most relevant node to the query at each step. In contrast, the non-iterative approach aims to directly extract the subgraph that is collectively the most relevant. Our Subgraph Retrieval method draws inspiration from both of these strategies. We start by identifying the initial seed nodes and then construct the subgraph iteratively using embeddings. However, we further enhance this process by introducing an additional selection step: we calculate an overall relevance measure (also based on embeddings) and determine which subgraph is the most relevant as a whole.

In section 2.7.4, we then explored the techniques used to integrate these different retrieval approaches. Most research has primarily focused on a straightforward strategy: concatenating the contexts extracted from multiple retrieval methods and feeding them to the LLM. While this approach has been shown to be more effective than using Vector-RAG or Graph-RAG alone, it presents two significant limitations. First, it typically incorporates only two retrieval sources: introducing additional sources would lead to a substantial increase in the context size passed to the LLM, making it impractical and potentially reducing the model’s ability to generate accurate responses. Second, these methods assume that relevance signals from different retrieval sources contribute equally to determining the best answer, an assumption that does not necessarily hold in all cases.

More advanced retrieval strategies often adopt a pipeline-based approach, where multiple retrieval methods are applied in sequence. A common strategy involves first performing Node Retrieval to identify a relevant subset of the KG, followed by Subgraph Retrieval or a specialized Search Module to further refine the results. While these approaches enhance retrieval efficiency, they fundamentally differ from our method. Existing techniques can be considered hybrid in the sense that they leverage multiple retrieval mechanisms, but they do not integrate different techniques to compute a comprehensive relevance measure. Instead, they simply chain multiple retrieval steps together. In essence, this is similar to what Node Retrieval already does: combining vector-based retrieval with the graph structure to extract an entire neighborhood, without explicitly synthesizing multiple relevance signals into a unified measure. Our approach, in contrast, aims to independently extract relevance signals from different retrieval methods and

fuse them in a non-trivial manner to determine the most relevant passages.

In conclusion, the state of the art still lacks an approach for representing a collection of unstructured documents in a structure that preserves all three of its original informational dimensions: content, organization, and semantic relationships between referenced entities. This gap directly impacts retrieval methods, which are inevitably dictated by the underlying KB they rely on.

Furthermore, there is no hybrid retrieval method capable of integrating relevance signals from multiple retrieval approaches into a single, comprehensive relevance score. Existing methods typically concatenate the retrieved contexts from different retrieval strategies or arrange them sequentially in a pipeline, where each stage filters the chunks extracted by the previous method. This means that none of these approaches have explicitly addressed the problem of determining the relative importance of their underlying retrieval strategies, nor have they developed a mechanism to reflect these varying weights in a final, unified relevance score.

Chapter 3

Methodology

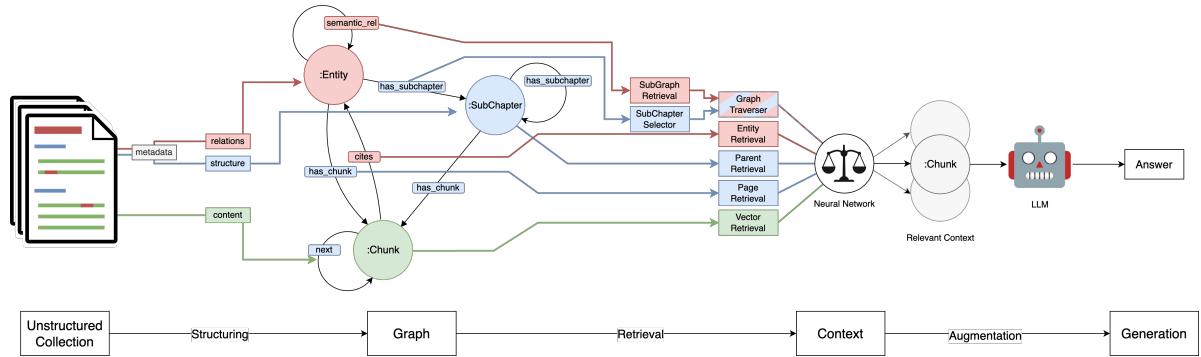


Figure 3.1: Complete Approach

In this section, we describe in detail our entire approach, represented in figure 3.1. In section 3.1 we start off by defining how we structure our initial collection of unstructured documents into the KG. We then explain in section 3.2 how each individual retrieval method is tasked with the identification of relevant chunks by leveraging different informational dimensions of the original document, and how they are consequently fused together into a single, comprehensive, relevance score.

3.1 Document Structuring

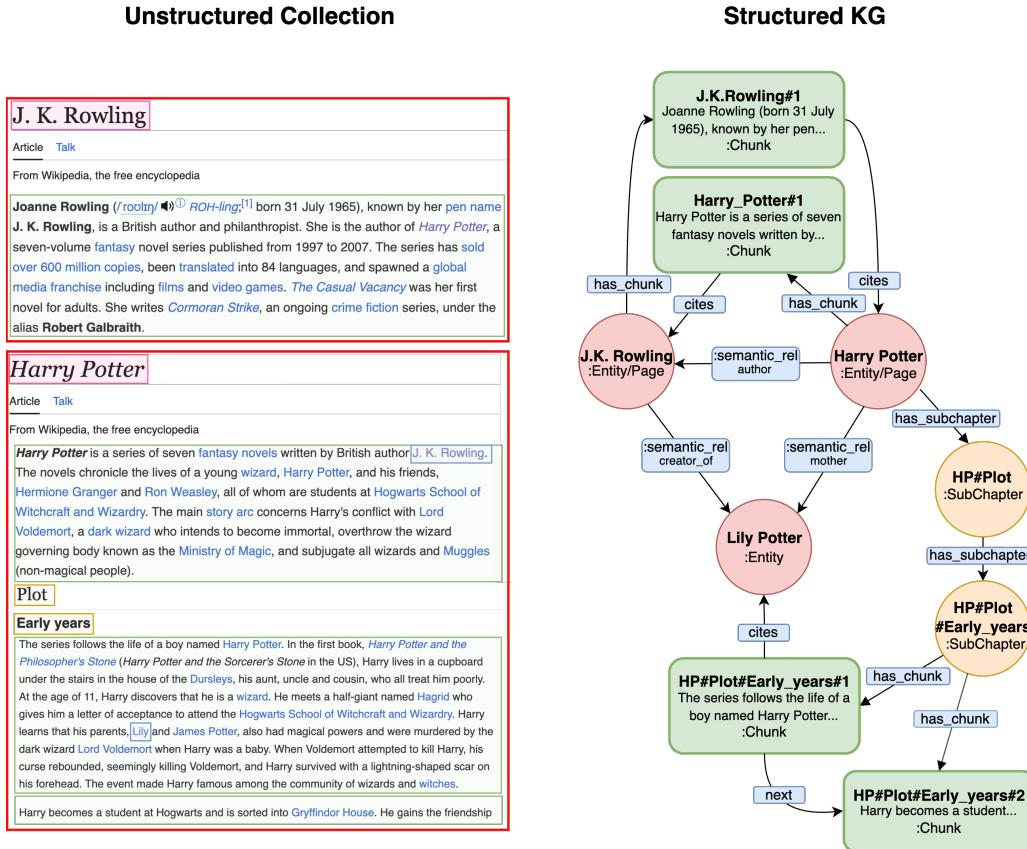


Figure 3.2: Example of the proposed solution for structuring a collection in the KG

This section focuses on the first step of our entire approach: the transformation of unstructured documents into a structured representation within our KG. As illustrated in figure 1.2 of section 1.1.1, and further exemplified through an example in Figure 3.2, starting from an original unstructured collection of encyclopedic documents, we aim at representing the information present in the document into a more structured KB.

The objective of the new structure represented in figure 3.3 and described hereafter is that of explicitly representing not only the content of the document, but also its organization and the semantic relationship between the entities mentioned in it.

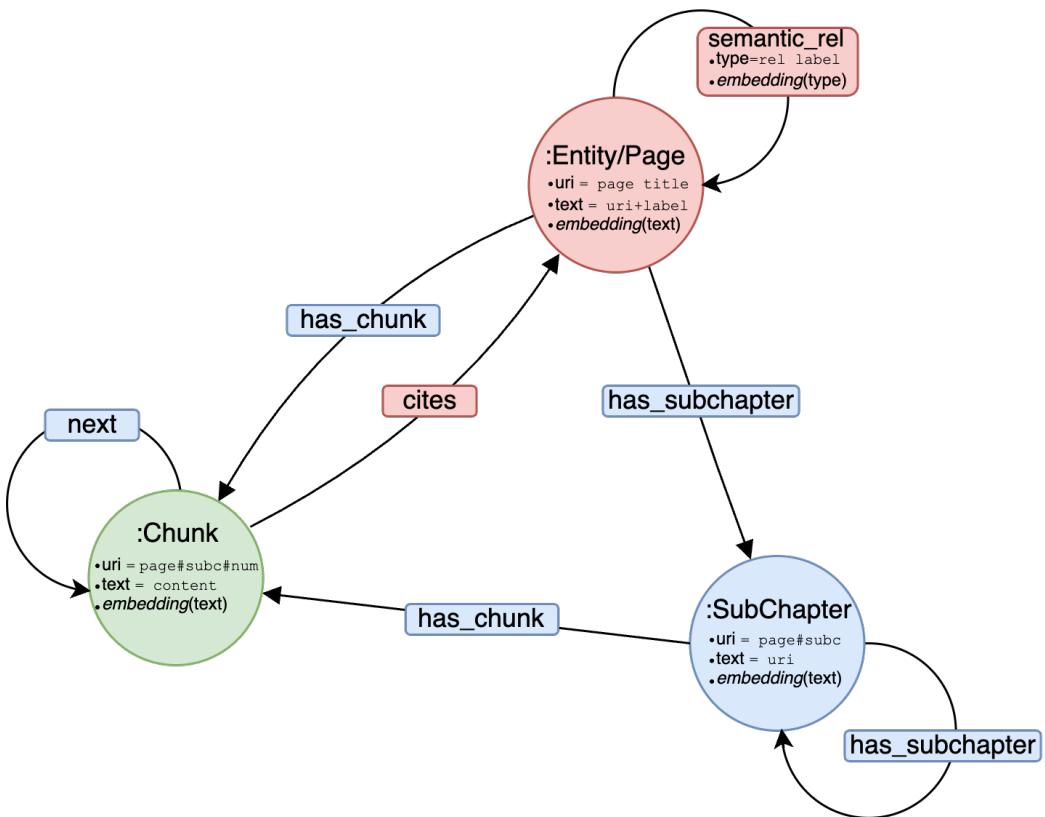


Figure 3.3: Graph Structure

1. **Entity** and **Page** nodes. As anticipated in 2.6, a prerogative of our approach is that of working with a collection of “encyclopedic” documents, defined as any collection in which the user’s abstraction of an “entity” directly corresponds to the concept of a document itself. We represent them in our KB as **Entity** and **Page** nodes.

Note that we will use the term “document” and “page” interchangeably, as each of our starting document is a Wikipedia page. Naturally, since the concepts of entity and document coincide, their corresponding nodes overlap too. More precisely, since our approach also allows for the inclusion of external entities that do not have a corresponding descriptive document within the collection, every **Page** node is also an **Entity** node, but not all **Entity** nodes are a **Page**, as there are entities that have been mentioned and extracted from the documents but that do not have a descriptive document themselves.

To maintain clarity, we will use the term “Page” when referring to nodes from a structural perspective, specifically in the context of subchapters and the chunks they contain. Conversely, we will use the term “Entity”

when discussing nodes from a semantic perspective, such as their relationships with other entities or their citations within external chunks.

- (a) **Identification** Each Entity node is uniquely identified by its `uri` property, which corresponds to the name of the entity itself. Given the overlap between entities and pages, this means that Page nodes are likewise identified by their titles, that is the name of the entity that the page describes.
- (b) **Properties** Each Page node has a `text` property derived from its `uri` property. The `text` property is further augmented by labelling the entity, using in our case DBpedia’s entity type, and appending the label to the property inside parenthesis. Since the `text` property is the one that is going to be used to create the embedding of the node, this expedient will help to better find and disambiguate the most relevant nodes for the question. For instance, if the question states “Who wrote [name of a poem]?", then the addition of the word “writer” in the embedding of an entity could facilitate the retrieval module to better pinpoint the correct answer. While this approach may yield marginal improvements in widely accessible knowledge domains like in this experiment, it holds particular promise for domain-specific knowledge bases, where entities may not have been encountered by the LLM during training.
Each Page node has an `embedding` property corresponding to the embedding of the `text` property.
- (c) **Relations** Entity nodes are connected together by a `semantic_rel` relations that makes explicit the relation between them. To mimic a practical approach, only entities mentioned within the same document will be connected together, as there would not be a way to extract from the original collection an information that ties two entities that are never mentioned together. To group these relations together, we give them all the same label (`semantic_rel`) and then disambiguate them by inserting the relation’s label into the property `type`. For each of this relations, a symmetric relation with the same label is build, connecting the object to the subject by a relations whose `type` property is equal to `is_[text of the original relation]_of`.
Each `semantic_rel` relation has an `embedding` property corresponding to the embedding of the `type` property.
- (d) **Captured Dimension** Entity nodes, along with their semantic relationship and the “cites” relation introduced later, capture from

the original document semantic information relative to the *mentioned entities and the relations that connects them*. As many of the research we reviewed in section 2, this portion represent the content of the document as a traditional KG composed of triplets of atomic entities.

2. SubChapter nodes.

All the sub-chapters of the page are extracted and stored in a SubChapter node.

- (a) **Identification** Each SubChapter node is identified via an `uri` property, consisting of the title of the starting page concatenated with the titles of all the sub-chapters traversed to come to the sub-chapter in question, joined via a hashtag (#) symbol. For example, the `uri` of the chapter 2.1 will be composed of the title of the page plus the title of the second sub-chapter plus the title of the sub-chapter 2.1 itself, all concatenated via a hashtag symbol.
- (b) **Properties** Each SubChapter node contains a `text` property that mirrors its `uri` property. While this redundancy might seem inefficient in terms of memory usage, it is necessary because all nodes are embedded based on their `text` property.
- (c) **Relations** Each SubChapter node is connected via the `is_subchapter_of` relation to *all* of their parent sub-chapters and the parent page. Its symmetric relation `has_subchapter`, though, is not transitive, meaning that it only connects a parent sub-chapter with its direct sub-chapters children. This choice will prove useful in later chapters, when the RAG system will be developed.
Each SubChapter node has an `embedding` property corresponding to the embedding of the `text` property.
- (d) **Captured Dimension** Subchapter nodes, along with their relationship, capture from the original document information relative to the *structure and organization* of the textual content. This portion enables the retrieval algorithm to move within the content of the document like a human would: relying on position and chapter's titles.

3. Chunk nodes.

All of the textual content of the page is extracted and divided into chunks, each of which constitutes a Chunk node.

- (a) **Identification** Each Chunk node is identified via an `uri` property, consisting of the `uri` of its parent sub-chapter plus an ordinary number representing the chunk's position in the chapter. So, for example, the `uri` of the second chunk of the chapter 2.1 will be composed of: the title of the parent page, the title of the second sub-chapter, the title of the sub-chapter 2.1 itself and the number two, all concatenated with a hashtag symbol.
- (b) **Properties** Each Chunk node has a `text` property consisting of the textual content of the chunk itself.
- (c) **Relations** Each Chunk node is connected back to *all* of its parent sub-chapters and the parent page via the relation `is_chunk_of`. Similarly to the `has_subchapter` relation, though, the symmetric relationship `has_chunk` is not transitive, this means that it only connects a sub-chapter with its direct chunk children.
 Each Chunk node also has a `next` relation that connects it with the subsequent chunk of the same subchapter, and a symmetric `previous` relation that links it with the previous one.
 Finally, each Chunk node is linked to every Entity node it references through a `cites` relationship. Conversely, the symmetric `cited-in` relationship connects each Entity node to the Chunk nodes in which it is mentioned.
 Each Chunk node has an `embedding` property corresponding to the embedding of the `text` property.
- (d) **Captured Dimension** Chunk nodes are designed solely to capture the *content* of the original document. This portion of the knowledge graph represents its core structure: the equivalent of what any standard vector database would contain.

3.2 Retrieval

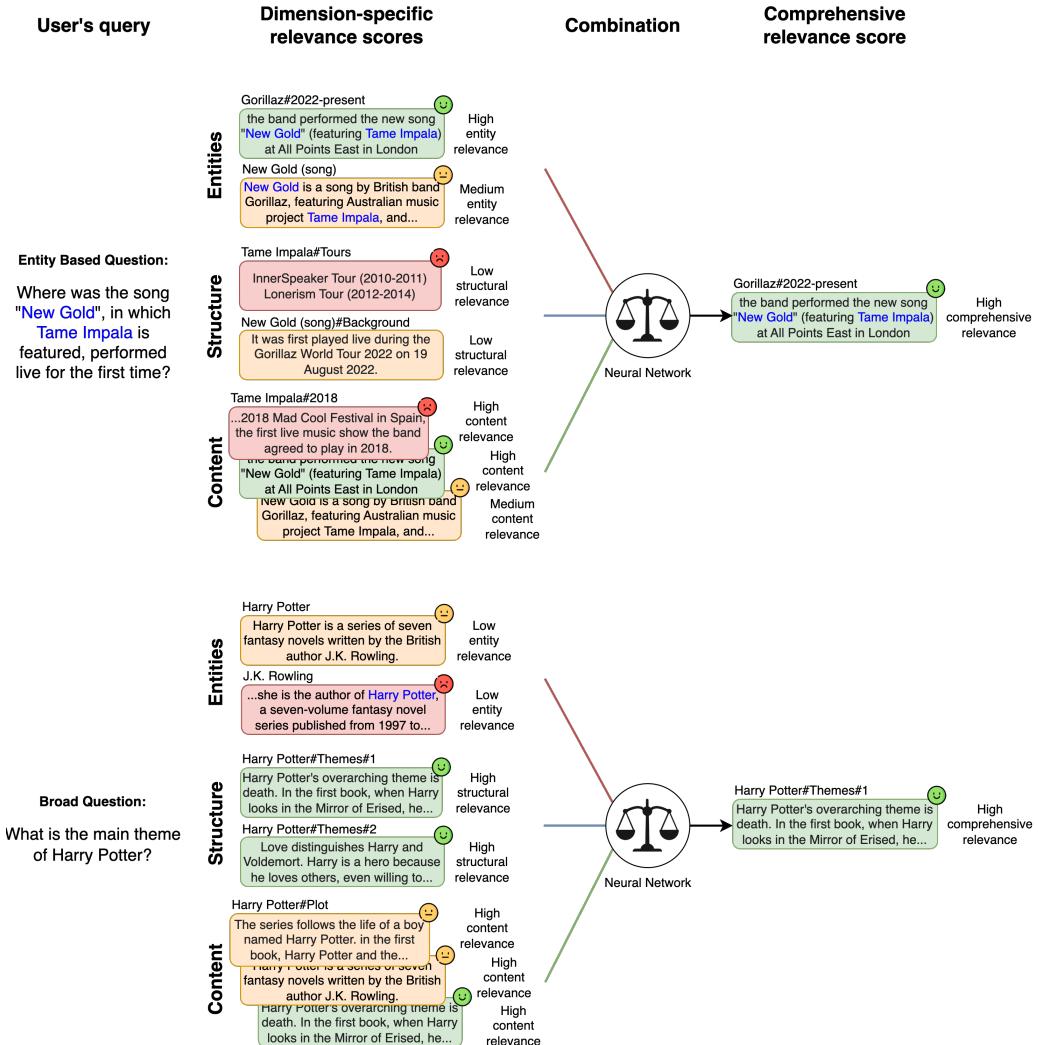


Figure 3.4: Example of the proposed solution for chunks retrieval

In this section we describe our methodology for addressing our second research question: how to design a retrieval component that leverages not only the content of a document but also the enriched information incorporated into our KB.

In section 3.2 we begin by presenting the various individual retrieval approaches employed in our approach one by one. In section 3.2.2, we then describe our strategy for integrating the relevance signals obtained from these methods into a final global decision. Rather than assigning equal weights to

each source, we develop our strategy by focusing on merging the different approaches in a way that reflects the actual importance of each signal in identifying the most relevant passages.

The intuition behind our approach is illustrated in Figure 3.4. This figure builds on the same questions presented in Figure 1.3 (Section 1.1.2), which we used to highlight how retrieval methods that rely on a single informational dimension inevitably fail to retrieve the most relevant chunks. In contrast, we aim to demonstrate that, by integrating multiple informational dimensions to identify the overall best chunks, our approach can achieve better results in most types of questions.

For each question, each retrieval method is tasked with identifying the most relevant chunks based on its preferred informational dimension. Additionally, each method is required to assign a relevance score to each chunk according to its specific dimension. By combining these individual scores, our approach constructs an overall relevance score, allowing it to determine the most suitable chunks across all dimensions. This ensures a more comprehensive and effective retrieval process, overcoming the limitations of single-dimensional approaches.

3.2.1 Retrieval Approaches

Name	Question Addressed	N. of retrieved nodes	Chunks Retrieved	Score
Vector Retrieval	How relevant is the chunk's content ?	10 Chunk nodes	Retrieves chunk directly.	Cosine similarity between the chunk and the query.
Page Retrieval	How relevant is the document to which the chunk belongs?	3 Page nodes	Retrieves all the chunks of the relevant pages.	Cosine similarity between the parent page's title and the query.
Section Retrieval	How relevant is the section to which the chunk belongs?	5 SubChapter or Page nodes	Retrieves all the chunk of relevant sections.	Cosine similarity between the parent section's title and the query.
Entity Retrieval	How relevant are the entities that the chunk cites?	5 Entity nodes	Retrieves all the chunks that cite one of the retrieved entities.	Max cosine similarity between the cited entity's name and the query.
Path Retrieval	Is the chunk linked to the relevant KG path ?	1 KG-path	Retrieves all the chunks relevant to the extracted path.	Binary: 1 to all chunks retrieved, 0 otherwise.

Table 3.1: Summary table of the retrieval methods employed.

In this section we describe each individual retrieval method focusing on the different aspects of the KG they focus on to assign their relevance scores. We summarize this information in table 3.1.

Vector-based Retrieval with Embeddings

*How relevant is the chunk's **content**?*

The most common and easy way to set up a RAG system is by implementing a vector-based retrieval with embeddings. The same model used to embed the text property of the nodes of the KG is used to embed the user's query. The resulting query-vector is then compared to the vectors of all the Chunk nodes via cosine similarity, i.e. the cosine of the angle formed by the

two vectors. This measure determines whether two vectors are pointing in roughly the same direction and is computed as follows:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Finally, the top k chunks with highest cosine similarity are retrieved.

The Vector-based retrieval is only applied to chunk nodes and therefore provides relevance information only from the perspective of the *content* of each paragraph.

This methods retrieves the top 10 relevant chunks, and assign them a score equal to their cosine similarity with the query's embedding.

Page Retrieval

*How relevant is the **document** to which the chunk belongs?*

As presented in detail in section 2.7.1, Node Retrieval consists in the retrieval of the whole neighborhood of a set of relevant nodes. This stems from the assumption that the neighbor of a relevant node will be relevant as well, as it provides additional context into the node itself. Page Retrieval consists in identifying the three most relevant **Page** nodes through cosine similarity, confronting the query with the **text** property of each page node. Then, **is_chunk_of** relations are harnessed to retrieve all text chunks that belong to that page.

This methods assigns to each retrieved chunk a score equal to the cosine similarity between the query and the parent page's title.

This approach leverages the *organization* of the text into documents to identify the most relevant chunks.

Section Retrieval

*How relevant is the **section** to which the chunk belongs?*

Section Retrieval is a second Node Retrieval approach that operates similarly to Page Retrieval. However, instead of focusing solely on the most relevant page, this method also considers **SubChapter** nodes. Additionally, rather than extracting all chunks connected directly or indirectly (through the transitive **is_chunk_of** relation), it only retrieves those directly linked

to the identified nodes (through the non-transitive `has_chunk` relation). In other words, this retrieval method has a narrower focus, as it favors the retrieval of chapters and sections rather than entire pages.

This method retrieves the top 5 subchapter and page nodes, and assigns to each connected chunk a score equal to the cosine similarity between the query and the parent chapter.

This approach completes the previous one by leveraging yet again the *organization* of the text, this time into sections, to identify the most relevant chunks.

Entity Retrieval

How relevant are the `entities` that the chunk cites?

Entity Retrieval is yet another Node Retrieval method that extracts the top 5 `Entity` nodes whose embeddings are most similar to the query. It then considers all `Chunk` nodes that reference these entities, leveraging the `cites` relationships present in the graph.

This method assigns to each retrieved chunk a score equal to the cosine similarity between the query and the cited entity's `text` property. If one chunk cites more than one relevant entity, only the highest relevance score is retained.

This approach shifts focus to *semantic metadata*, particularly emphasizing entities and the chunks that reference them.

Path Retrieval

Is the chunk linked to the relevant KG path?

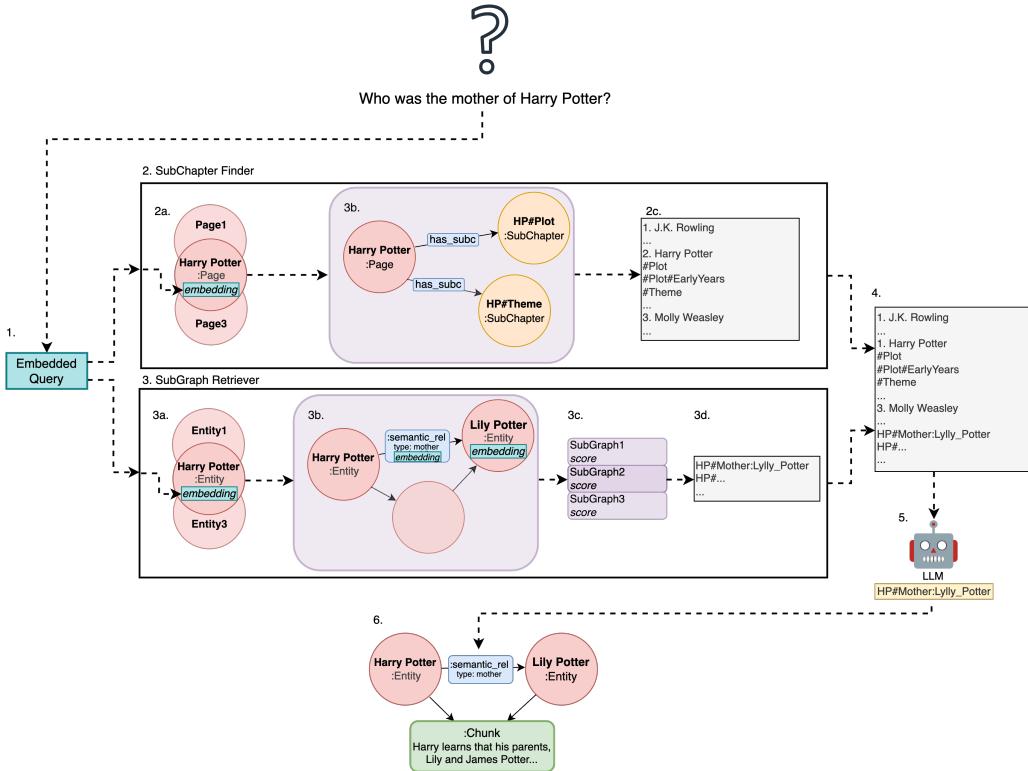


Figure 3.5: Path Retrieval

As mentioned in 2.7.2, the employment of an LLM to translate natural language queries into Cypher queries did not yield satisfactory results due to the specific structure of our KB. However, drawing inspiration from KnowledGPT[8], we adapted this approach to better suit our specific needs.

The path is retrieved thanks to the employment of a “Graph Traverser”, which works by employing an LLM to select, out of a list of possibly relevant paths, the one that is most relevant to the question. In particular the traverser is able to select from a list coming from two types of relations: the **has_subchapter** relations, selected by the “SubChapter Finder”, and the **semantic_rel** relations, selected by a “SubGraph Retriever”.

The employment of a SubGraph Retriever, as those reviewed in section 2.7.3, can introduce additional relevance signals into the model by incorporating entities and the semantic relationships that connect them. Furthermore, the traversal-based approach employed in this method offers a dis-

tinct perspective that can be particularly valuable when handling Multi-Hop questions. In fact, up to this point, we have primarily focused on retrieval methods that aim to directly pinpoint the correct answer or its immediate context. In contrast, subgraph retrieval is the only approach that actively leverages relationships and navigates through them to locate the answer.

Hereafter is provided the entire approach of the Graph Traverser, also represented in 3.5:

1. Query Embedding

The user’s query is embedded.

2. SubChapter Finder

(a) Page Retrieval

The three most relevant Page nodes are identified based on cosine similarity.

(b) Subchapter Extraction

All the subchapters of the retrieved pages are extracted.

(c) Formatted Text Construction

The chapters are organized into a properly formatted text suitable for the next stage of LLM injection:

```
1.Title of page 1
#Title_subchapter1
#Title_subchapter1#Title_subchapter1.1
#Title_subchapter1#Title_subchapter1.2
#Title_subchapter2
ecc.
2.Title of page 2
ecc.
```

Listing 3.1: SubChapter formatting for SubChapter Finder

3. Subgraph Retrieval

(a) Entity retrieval

The three most relevant Entity nodes are identified. They serve as the foundation for subgraph construction.

(b) Iterative expansion

From these starting nodes, the algorithm expands outward by exploring their neighborhoods and iteratively retrieving the most relevant connected nodes. At each iteration, the selection of the next node is guided by a comparison embedding, computed as the average of the node’s own embedding and the embeddings of all relationships traversed to reach it from the seed node. The node

within the current neighborhood whose comparison embedding has the highest cosine similarity to the query embedding is added to the subgraph. Once a new node is incorporated, the considered neighborhood expands to include all of its neighbors as well. This iterative process continues until the subgraph reaches a predefined size (5).

(c) **Global selection**

Each subgraph is assigned an overall relevance score by averaging the individual scores accumulated during its expansion. The subgraph with the highest score is in this way selected as the final result.

(d) **Formatted text construction**

The relations are organized into a properly formatted text suitable for the next stage of LLM injection:

```
NodeA#rel_type:NodeB  
NodeA#rel_type:NodeC  
NodeB#rel_type:NodeD
```

4. Context concatenation

The output of the SubChapter Finder and the SubGraph Retriever are concatenated.

5. LLM selection

The LLM is prompted with the following System Prompt:

```
You are going to be provided with a context consisting of  
multiple Wikipedia pages, along with all of their  
chapters and subchapters.  
If existent, return the name of the page + the name of the  
chapter or subchapter in which the answer to the user  
's question can be found.  
If the question is about something generic of the topic of  
the page, return the title of the page.  
If the question is specific and none of the pages'  
chapters and subchapters fits it precisely, return 'ND'  
'.  
If the answer could be found in multiple pages or  
subchapters, or you are undecided between multiple  
options, respond with a list of ALL the identifiers of  
the relevant pages or subchapters, separated via a  
semicolon.  
If the question needed multiple subchapters to be answered  
, respond with a list of ALL the identifiers of the  
subchapters used to answer the question, separated via  
a semicolon.  
In any case, return the answer without providing any  
comments.
```

Listing 3.2: System Prompt for Graph Traverser

6. Chunk Retrieval

Now that we have identified the most relevant subchapter or graph path we want to use this information to retrieve the most relevant chunks. If the LLM selects a subchapter, then all chunks directly belonging to that chapter are added to the relevant chunk list. Instead, if the LLM selects a KG-path, all nodes involved in that path are extracted. If a chunk that cites all of the nodes in the path exists, it is extracted as the main relevant chunk. Otherwise, if at least one of the nodes is a Page node and a chunk within that page citing all the other nodes exists, that chunk is selected. If this condition is not met and more than two nodes are involved in the path, all chunks citing at least two of them are included. In cases where none of the above criteria apply, any chunk that cites at least one of the nodes is added to the list.

The method assigns a score of 1 to all the Chunk nodes retrieved, and a score of 0 otherwise.

This method is dual-faceted: its SubChapter Finder component focuses on the *structural organization* of the original document, while its SubGraph Retrieval component emphasizes the *semantic dimension*, leveraging entities and the relationships that interconnect them.

3.2.2 Hybrid Retrieval

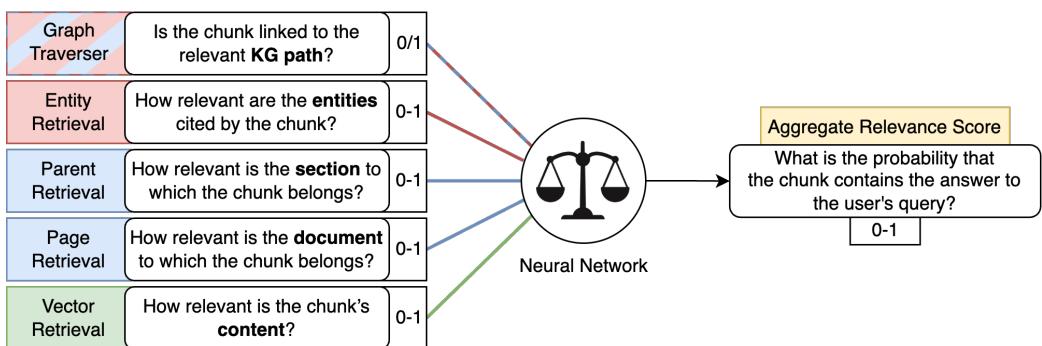


Figure 3.6: Hybrid Retrieval

Although, as we'll see in section 4, none of the retrieval methods individually outperforms traditional document retrieval, each one of them leverages

different aspects of the graph to capture distinct facets of the answer. Drawing inspiration from ensemble learning, this section defines an approach that integrates these various relevance scores into a single, unified relevance measure, ensuring that their complementary strengths are leveraged to minimize errors and enhance accuracy. The main focus is to build this hybrid retrieval so that it weights the starting relevance scores proportionally to the actual importance that each initial method has in identifying the most relevant passages.

We model our hybrid retriever as a Feed Forward Neural Network (FFNN). The network gets in input, for a specific Chunk node c , its relevance scores for all of the native retrieval approaches ($[c_v, c_{page}, c_{parent}, c_{entity}, c_{pr}]$). The FFNN is trained to classify the node c into two classes: relevant (1) or not relevant (0). This means that, during inference, the output of the network $\sigma(f(c))$ for each node c is equal to the probability of that node to be relevant: $\sigma(f(c)) = P(c_{relevant} = 1|c)$.

The FFNN consists of three fully connected hidden layers, each followed by batch normalization and dropout regularization to enhance generalization and prevent overfitting. Specifically, the architecture is structured as follows:

- Input layer: Receives the five-dimensional feature vector corresponding to the chunk’s retrieval scores.
- Hidden layers:
 - First hidden layer: 254 neurons, ReLU activation, L2 regularization ($\lambda = 0.01$), followed by batch normalization and a 20% dropout rate.
 - Second hidden layer: 128 neurons, ReLU activation, L2 regularization ($\lambda = 0.01$), batch normalization, and a 30% dropout rate.
 - Third hidden layer: 64 neurons, ReLU activation, L2 regularization ($\lambda = 0.01$), batch normalization, and a 40% dropout rate.
- Output layer: A single neuron with a sigmoid activation function, producing a probability score indicating the relevance of the chunk.

The network is trained using the Adam optimizer with a learning rate of 0.001 and binary cross-entropy as the loss function. To improve convergence and prevent overfitting, we incorporate early stopping (with patience set to 10 epochs) and learning rate reduction on plateau (reducing the learning rate by a factor of 0.2 after 5 epochs without improvement, with a minimum learning rate of 0.0001). The model is trained for up to 100 epochs with a batch size of 64.

While the actual architecture of the network is relatively straightforward, the overall approach must incorporate some level of ingenuity to address three main design challenges that arise when shaping a tool of this kind. The effectiveness of the solutions described hereafter will be demonstrated in section 4.5.2.

Local Standardization

The first challenge arises from the nature of the task itself. Instead of a straightforward classification problem that determines whether each chunk is relevant or not, the objective is to select the top- k most relevant chunks from the subset $\mathcal{C}_{\text{relevant}}$, which consists of all chunks identified as relevant by at least one method. The output of the neural network remains a probability (ranging from 0 to 1) indicating the likelihood that a given chunk contains the answer to the user’s query. However, this probability is not computed in isolation; rather, it is derived by comparing the relevance scores of individual chunks within the same subset, rather than against all extracted chunks across the entire dataset of queries.

Before training a neural network, normalization is commonly applied to the dataset to enhance model convergence and stability by ensuring that input features are on a comparable scale. In our case, it would be beneficial to standardize the continuous relevance scores produced by each individual retrieval method. However, if we would apply global standardization, i.e. scaling each continuous variable relative to all observations in the dataset, the model would inherently compare each instance to the entire dataset rather than to its specific group. In this context, if an observation receives the highest relevance scores within its question-specific set, but relatively low scores compared to the overall average across all questions, global standardization would significantly lower these values. Feeding these reduced scores into the model could result in the misclassification of the observation as non-relevant.

To mitigate this issue, we propose standardizing the continuous variables of each observation using only the values within its respective group. This adjustment ensures that the model implicitly compares each observation to others within the same set. As a result, the observations that retain the highest values when passed to the model are not necessarily those that rank highest across the entire dataset but rather those that stand out within their specific group.

To achieve this, instead of computing the mean and variance of the relevance scores for each retrieval method across all chunks in the dataset and using these values to standardize the relevance scores globally, we adopt a localized approach. Specifically, for each question in our QA dataset, we con-

sider the set of chunks retrieved by at least one retrieval method, denoted as $\mathcal{C}_{\text{relevant}}$. Within this set, we compute the mean and variance of the relevance scores for each retrieval method using only the chunks in the group. These values are then used to standardize the relevance scores within the group itself, ensuring that the scaling is performed relative to the specific query context rather than the entire dataset.

Handling Missing Values

A second challenge arises from the inevitable sparsity of the dataset on which we want to train our model. Since $\mathcal{C}_{\text{relevant}}$ represents the subset of chunks retrieved by at least one retrieval method, most observations within it will have several missing scores, as they were selected by only one or a few of these methods. It is therefore necessary to determine how to assign values to these unobserved variables.

Since the cosine similarities of retrieved chunks in the train set typically fall within a narrow range (0.85–0.90), replacing this null values with 0 (before standardization) would have resulted in non-retrieved observations receiving extremely low standardized values, while the retrieved ones would have been confined to a very small and distant range. Since these observations were retrieved by at least one method, they are inherently relevant to the question. Even if they were not in the top n retrieved chunks for that specific retrieval approach, it is still implausible that they would have had a relevance score of 0. Instead, it is much more likely that they would have received a relatively high cosine similarity, albeit lower than the set’s actual minimum for that retrieval method. Rather than assigning a score of 0, representing a total insignificance of the observation to that specific retrieval approach, it is much more reasonable to assign an arbitrarily high, but still lower than every other observation, relevance value. Since the overall lowest score was of 0.81, we opted for replacing every missing value with a standard relevance score of 0.75, included before standardization.

Active Sampling

One aspect of the hybrid retrieval training method that we found concerning was the subsampling phase. A traditional subsampling method involves retaining all observations from the minority class (in our case, $c_{\text{relevant}} = 1$) and randomly sampling an equal number of observations from the majority class ($c_{\text{relevant}} = 0$). However, purely random sampling does not guarantee the quality of the latter class of observations. For each question, a significant portion of the chunks listed in the dataset may have been identified

as relevant by only one method. A chunk is more likely to be genuinely relevant if multiple methods have flagged it as such. This implies that, for any given question, many chunks labeled as $c_{relevant} = 0$ will be “clearly” irrelevant, providing little challenge for the model during training. Random sampling increases the likelihood of including these “easy negatives”, which do not contribute meaningfully to the learning process.

A more effective sampling strategy might involve identifying “ambiguous” observations, i.e. those chunks detected as relevant by multiple methods, and using these to train the model. By doing so, we would train the network on more complex data, with the expectation that the model would learn deeper patterns. The hope is that a model trained to correctly distinguish more intricate data will also perform well on simpler observations. The strategy of training a model with harder examples is called *curriculum learning*[88], a technique where models are first trained on simpler examples before gradually being exposed to more complex and ambiguous cases, improving their generalization ability.

One approach to identify more complex examples is to leverage the scores produced by a second neural network, pre-trained on data sampled using traditional methods. This technique is formally known as *hard-negative mining*, where hard negatives refer in this case to observations that, despite being truly negative, were mistakenly assigned a high probability of belonging to the positive class by the first neural network. By incorporating these challenging examples into the training process, the model is encouraged to learn more complex reasoning paths, ultimately improving its ability to distinguish between relevant and non-relevant chunks. Therefore, we propose the following training pipeline:

1. Apply traditional sampling.
2. Train a first FFNN on the sampled data.
3. Utilize the network to assign to each chunk a probability of belonging to the positive class.
4. From the original dataset, refine the $\mathcal{C}_{relevant}$ set of each question by eliminating the negative observations that fall within the lowest 90% probability scores. This ensures that only the most promising candidates are retained for further analysis, thereby enhancing the overall quality and relevance of the dataset.
5. Apply traditional sampling on the refined dataset.
6. Train a second FFNN on the more complex dataset.

Chapter 4

Experiments

Now that we have established the methodology for addressing the objectives of this thesis, the next step is to define the dataset that will be used to conduct the experiments and evaluate our approach.

4.1 Dataset

Our approach is tested on two different QA datasets: a 20% portion of the development set of NQ, designed for single-hop questions, and a synthetically generated multi-hop dataset composed of 100 questions. In particular, we use the Wikipedia pages referenced in the dev set of NQ to build our KG by developing a “Wikipedia Parser”. The synthetic multi-hop dataset is then generated from this same KG.

4.1.1 Natural Questions Dataset

For this experiment, we use the development set of Google’s Natural Questions dataset (NQ)[22] (hereafter referred to as NQ-dev) to train and evaluate our Hybrid Retrieval system. The NQ-dev set consists of approximately 8,000 observations, each associated with a relevant Wikipedia page. All of these 8,000 pages are used to construct our KG, which serves as the retrieval source during both training and testing, for both single-hop and multi-hop queries. Instead, for what regards the questions, 5,223 of them actually have an annotated answer that we were able to trace back in our KB. These 5,223 questions are split into a training and testing set to ensure an unbiased evaluation. Specifically, 80% of the questions in NQ-dev (NQ-dev-80%) are used to train the Hybrid Retrieval system, while the remaining 20% (NQ-dev-20%) are exclusively reserved for testing, resulting in 4,179 training questions and

1,044 testing questions.

Google’s NQ was selected as the benchmark for the question-answering task due to several key advantages.

First of all, NQ leverages Wikipedia as its knowledge source, with answers appearing as text spans within Wikipedia pages. As we anticipated in 1.1.1, we select Wikipedia because it greatly simplifies the implementation of the experiment. Wikipedia inherently follows an encyclopedic structure, where each page is dedicated to a single topic, and every topic is clearly defined by its corresponding page. Additionally, references to other pages are explicitly marked with hyperlinks, making them easy to parse. While our approach is generalizable to any knowledge source with a similar structure, applying it to plain text would require prior processing steps such as NER, NEL, and Coreference Resolution. Since Wikipedia already highlights and links entities, we can bypass this preprocessing phase and focus directly on developing the KB structure and retrieval system. The same advantage applies to relationships between entities: thanks to parallel knowledge sources like DBpedia, we can retrieve the type of connection between two entities without relying on Relation Extraction models.

Secondly, to limit the size of the dataset to a manageable size, we need to define a subset of Wikipedia pages to form our initial unstructured KB. Since each question in the dataset specifies the Wikipedia page from which the answer is derived, we can efficiently create a subsample and restrict our database to only relevant pages (the dev set).

Finally, it is a high-quality, widely-used dataset, which has been extensively utilized in the literature on QA systems, underscoring its credibility and reliability. Furthermore, its questions come from real user interactions and therefore provide an evaluation on data resembling production environments.

Knowledge Base Construction

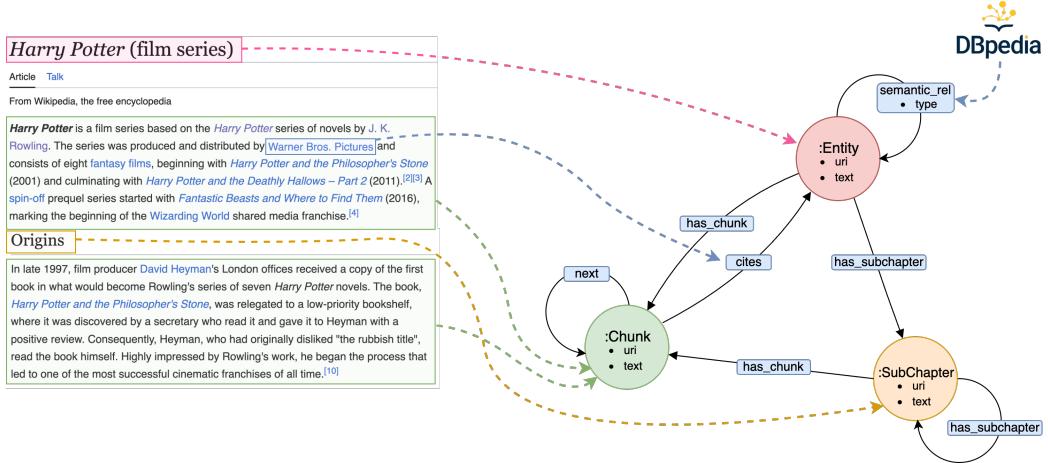


Figure 4.1: Wikipedia Parser

To transform the collection of unstructured documents (NQ-dev) into the organized Knowledge structure we defined in 3.1 we create a custom Wikipedia Parser, whose working is exemplified in figure 4.1. The parser is designed to work directly with the Wikipedia pages provided in the dataset, although a version that uses the provided title to download the corresponding Wikipedia page through its API¹ also exists. The first approach is preferred to avoid delays associated with live page downloads and to ensure consistency with the dataset's original version of Wikipedia pages.

The resulting HTML of each page is then parsed using BeautifulSoup². First of all, the title of the pages is extracted to create the corresponding Page node. Then, all subchapters are extracted to create the corresponding SubChapter nodes, and are linked to the parent page. For each subchapter, its content is extracted and divided into chunks leveraging Wikipedia's double new-line formatting. Each of these chunks constitutes a Chunk node, which is connected back to the parent subchapter, the parent page and to the adjacent chunks. The chunk parsing mechanism is enhanced to include a broader range of HTML elements beyond just paragraphs, such as tables and lists. This refinement is critical, as a significant portion of the answers in the NQ dataset are located in these elements, and failing to include them would adversely impact the results. For each chunk, the parser extracts all the referenced pages through Wikipedia's hyperlinks. If the extracted page is

¹www.mediawiki.org/wiki/API

²crummy.com/software/BeautifulSoup/bs4/doc/

not already in the KB, then an Entity node is created with its title and linked to the chunk with the “cites” relation. If the extracted page is already in the KB, then it is identified and linked to the chunk with the same relation.

After this first parsing of the entire collection, the parser identifies, for each extracted entity, the corresponding DBpedia entity thanks to the “is-PrimaryTopicOf” property. It then proceeds to transfer to the graph all the relations that connect two entities that are cited in the same page. These relations are all stored in the new database with the type `semantic_rel` and hold the original type in the property `type`.

For each question, in addition to the page in which the answer is stored, the NQ dataset stores the answer as a text span of the page itself. Each annotator is asked to outline a “long answer” (i.e. a whole chunk of the page) and, when possible, a “short answer” (i.e. the shortest possible span of text). Preserving this information is crucial for the evaluation of the retrieval systems. To do so, a new function is implemented into the parser to extract textual answers based on the spans provided in the “long answer” annotations, identify all chunk nodes linked to the corresponding Wikipedia page, and search within each chunk for potential matches. Matched chunks are annotated with an `is_answer_of` property that is set equal to the respective question. Notably, as some questions have multiple valid answers, all chunks containing these answers are appropriately identified and annotated.

The graph thus created consists of:

- **600,411 nodes**, including:
 - 165,803 nodes of type :Chunk
 - 366,660 nodes of type :Entity
 - 4,897 nodes of type :Page
 - 63,051 nodes of type :SubChapter
- **2,329,701 relationships**, including:
 - 1,009,327 relationships of type [:cited_in]/[:cites]
 - 420,505 relationships of type [:has_chunk]/[:is_chunk_of]
 - 63,051 relationships of type [:has_subchapter]/[:is_subchapter_of]
 - 103,538 relationships of type [:next]/[:previous]
 - 733,280 relationships of type [:vector_rel]

The construction of the KG for the 5,223 questions in NQ-dev that actually contained an answer was completed in 36 minutes. The first 20 minutes were dedicated to parsing the Wikipedia pages, which involved decomposing them into their fundamental components (entities, subchapters, and text chunks), extracting all relevant relations, and storing this information in separate CSV files. Next, 12 minutes were required to filter the DBpedia dumps, extracting only the relations relevant to the construction of our KG. This process included mapping Wikipedia pages to their corresponding DBpedia entities via the “isPrimaryTopicOf” relation and leveraging this mapping to retrieve DBpedia labels and relations connecting extracted entities. Finally, the bulk import into Neo4j took the remaining 4 minutes. This last step also included additional operations, such as assigning the “Page” label to entities associated with chunks and subchapters, and removing relations between entities that did not co-occur within the same document.

Now that the database is constructed, the next step involves creating embeddings for all nodes in the graph. To achieve this, OpenAI’s ada-v2[44] embedding model is employed through Azure’s OpenAI API. The process of embedding initialization and indexing is carried out using the langchain_community library³, specifically the Neo4jVector class and its `from_existing_graph()` method. This function connects to the Neo4j instance, constructs an index if one is not already present, retrieves up to 1,000 nodes lacking embeddings, and passes their text property to the embedding method. Despite reasonable execution speeds, this resulted in the process often triggering rate limit error. To mitigate this issue, the graph indexing process was divided into distinct steps for nodes and relationships.

Entity and SubChapter nodes contain short token sequences, such as page titles or subchapter headings. Therefore, the rate limit is triggered because the algorithm reaches the limit of 1,400 requests-per-minute. The `from_existing_graph()` function is modified to address this limitation. First, the graph query limit is increased from 1,000 to 1,200 nodes, leaving 200 requests as a buffer. Then, the exact time of embedding completion is recorded, and subsequent requests are delayed until at least one minute has elapsed since the previous batch. This approach required approximately 426 minutes for Entity nodes indexing and 73 minutes for SubChapter nodes indexing.

For Chunk nodes, since they represent larger text sequences, the token-per-minute limit (240,000) is the primary bottleneck. To resolve this, the function is further adapted. After retrieving 1,200 nodes, a subset is selected based on cumulative token count, with a safety buffer of 20,000 tokens. The

³pypi.org/project/langchain-community/

number of tokens per node is calculated using the tiktoken⁴ library with the `o200k_base` encoding. Nodes exceeding the token limit are deferred to subsequent iterations, ensuring complete indexing across multiple batches. This step required 203 minutes to complete.

Lastly, DBpedia relationships are also embedded. This step is carried on with relative ease, since only 7,668 unique relationship types are present. The embedding process involves iterating over each type, indexing its textual representation, and applying the resulting embedding to all relationships of that type. This process took 43 minutes. While parallelizing requests could have improved efficiency, the relatively small number of relationships did not justify further optimization.

All the embeddings are then stored in the respective property “embedding”. Finally, vector indices were constructed in Neo4j.

4.1.2 Synthetic Multi-Hop Dataset

Google’s NQ dataset is great because its questions come from real user interactions and therefore provide an evaluation on data resembling production environments. Though, each query is precise and its answer can be found inside a single chunk. Even for the question where the annotators selected multiple chunks as answers, in fact, only one of them is needed to answer the question. The multitude of responses is often not related to the actual need for each chunk of being retrieved, rather to the fact that multiple passages of the pages repeat the same information. In QA systems, though, the ability to answer natural language questions that involve extracting and combining multiple pieces of information is crucial, and is extensively studied in the field of Multi-Hop QA.

To test the accuracy of the RAG system also on this type of question, a synthetic Multi-Hop dataset (NQ-MH-100) is built from the KG.

First, a random set of 300 observations is collected from Entity nodes that are cited in at least two Chunk nodes. In this way, each observation contains an entity node and a set of at least two chunk nodes. Each observation is then passed to an LLM (GPT 4o) to which it is asked to write a Multi-Hop question that needs at least two of the chunks to be answered. Moreover, the LLM is asked to provide the answer to the question itself and the chunks that need to be retrieved to answer the question. The complete prompt is reported below:

```
Imagine being a system made for building test sets.  
You are going to be given two or more chunks of text, and an entity  
recurring in each of these chunks.
```

⁴github.com/openai/tiktoken

Write a Multi-Hop question that needs at least two of the chunks to be answered AND its answer. Provide also the name of the chunks from which the question has been formulated.
 The provided entity shouldn't either be the subject of the question nor the answer, but should be a piece needed to answer the question.
 If no straightforward question can be extracted from the chunks, answer with "ND".
 Here are some examples:

```

INPUT:
- chunk1: "The flag of [NATION] is [COLORS of FLAG]."
- chunk2: "[NATION] won the [SOCCER TOURNAMENT]."
- entity: [NATION]
OUTPUT:
Question: What color is the flag of the nation who won the [SOCCER TOURNAMENT]?
Answer: The flag of the nation that won the [SOCCER TORUNAMENT], [NATION ], is [COLORS of FLAG].
Chunk used: chunk1, chunk2

INPUT:
- chunk1: "[PERSON1] is the sister of [PERSON2]"
- chunk2: "[PERSON2] is from [NATION]"
- chunk3: "[PERSON2] is the lead singer of [BAND]"
- entity: [PERSON2]
OUTPUT:
Question: Who is the sister of the lead singer of [BAND]?
Answer: The sister of the lead singer of [BAND], [PERSON2], is [PERSON1 ].
Chunk used: chunk1, chunk3

The question should not simply ask about information of the entity in the first and second chunk, but should ask for a link between them.  

Here are some example of a good and bad question:
INPUT:
- chunk1: "In 1996 [ACTOR] performs in [FILM]."
- chunk2: "[FILM] talks about [TOPIC]."
- entity: [FILM]
BAD question: Who performed in [FILM] and what is [FILM] about?  

GOOD question: What is the film [ACTOR] starred in 1996 about?

INPUT:
- chunk1: "[PERSON1] invented [INVENTION]."
- chunk2: "[PERSON1] suffered of [ILLNESS]."
- entity: [PERSON1]
BAD question: What did [PERSON1] invent and what illness was he suffering from?  

GOOD question: What disease did the inventor of [INVENTION] suffer from?  

You should make sure that the question has only one straightforward answer.  

Here are some examples.  

BAD question: Who is the actor who played Jack in one of the most influential movies of the 90's?  

There could be more than one actor that played the role of a character named Jack in an influential movie of the 90's.  

GOOD question: Who is the actor who played Jack in the movie that came out on February 11th 1996?  

Making sure there is only ONE character named Jack in the movies that came out on that date.  

If no GOOD question can be extracted from the context, answer with "ND"
  
```

''.

Listing 4.1: Synthetic Multi-Hop Dataset System Prompt

Then, the context is provided to the model in the Human Message in the same way as exemplified in the examples of the system prompt. Finally, the generated output is parsed to extract the question, the answer, and the chunks used to formulate the question.

In the second phase, after removing all observations that generated a response equal to ND, all the outputs of the model are passed again into the LLM with a different prompt. In it, the model is asked to act as an evaluator that gives each observation a score based on the quality of the question-answer pair. The complete prompt is reported below:

```
Imagine being an evaluator.  
You are going to be given a question-answer pair of a synthetic multihop  
question answering dataset.  
Give a complexive score from 1 to 10 to the observation.  
The score should be lower if:  
- It is not a Multi-Hop question: the questions asks something about an  
entity that is directly and explicitly mentioned in the questioned  
itself.  
- The question is excessively convoluted.  
- There could be more than one answer to the question.  
- There may be ambiguity in identifying the entities the question is  
referring to.  
- The question asks multiple different things.  
  
Write a VERY BRIEF evaluation of the observation based on this question.  
Based on this evalation, write "FINAL SCORE: " and assign the score to  
the observation.
```

Listing 4.2: Evaluator System Prompt

The observation is then passed to the model, and the generated answer is parsed to extract the score.

In the last step of the process, the observations are ordered by their score and are manually evaluated by a human which deletes the unsuitable ones. The human evaluation continues till a total of 100 suitable question-answer pairs is reached. This set, consisting of question, answers and the uris of the Chunk nodes needed to answer the question, composes the Multi-Hop evaluation dataset.

It is important to note that this Multi-Hop dataset is used exclusively during the testing phase. This means that the neural network is evaluated on Multi-Hop questions despite being trained solely on the non-Multi-Hop dataset. This experimental setup allows us to assess the model's ability to generalize to more complex, Multi-Hop queries without any prior exposure to them during training.

4.2 Baselines

To evaluate the performance of our Hybrid Retrieval method, we compare it against each of the individual retrieval techniques that contribute to its construction. Among these, our primary baseline is the Vector Retrieval, as it is the most commonly used retrieval method in QA and RAG systems due to its simplicity and widespread adoption.

We also introduce an additional retrieval method to compare our approach against another hybrid strategy. Inspired by the work of HybridRAG[83] and LangChain⁵ described in section 2.7.4, we introduce a baseline approach termed Naive Hybrid Retrieval. This method applies both Path Retrieval and Vector Retrieval in sequence, simply concatenating their results. Specifically, for each query, we first retrieve up to three chunks using the Path Retrieval. If more than three chunks are retrieved, only those with the highest vector similarity to the query are retained. We then employ Vector Retrieval to retrieve additional chunks, ensuring a total of five retrieved passages. This approach serves as a simplistic form of hybrid retrieval, where multiple methods are combined without any deeper integration of their relevance signals. By using this as a baseline, we aim to compare our approach against a straightforward hybrid strategy, highlighting the advantages of our more refined retrieval mechanism.

Since this method simply concatenates the results of two different retrieval approaches, we believe that evaluating its effectiveness with only three retrieved chunks would not be meaningful. Given that the chunks are sourced from two distinct methods, a smaller retrieval set would not adequately reflect the combined contributions of both techniques. This issue becomes even more pronounced when considering just a single retrieved chunk, where the hybrid nature of the approach would be effectively lost. For this reason, we retrieve a total of five chunks for the standard QA task, ensuring a more balanced evaluation of the method’s performance. For Multi-Hop QA, however, since a larger number of chunks is already required to reconstruct multi-step reasoning paths, we maintain evaluations at 4, 6, and 8 retrieved chunks, in line with the other retrieval techniques.

⁵blog.langchain.dev/enhancing-rag-based-applications-accuracy-by-constructing-and-leveraging-knowledge-graphs/

4.3 Evaluation of the Retrieval System

In this section, we evaluate our system from a retrieval perspective. Specifically, we assess the effectiveness of the retrieval process by leveraging the independent variable “is_answer”, which we have assigned to each chunk. By applying our retrieval pipeline, we measure the proportion of retrieved chunks that actually contain the correct answer to the given question. The analysis of how this retrieval approach impacts the overall quality of responses generated by the RAG system is instead deferred to Section 4.4.2.

The questions in the NQ-dev-20% dataset are flattened to create a training dataset where each observation corresponds to a chunk retrieved by at least one of the retrieval methods. This process results in a dataset of 50,167 observations, among which 6,245 chunks are labeled with $\text{is_answer} = 1$, indicating that they contain the correct answer. To address class imbalance, active sampling is applied to reduce the number of $\text{is_answer} = 0$ observations to match the 6,245 positive cases. This results in a balanced training dataset of 12,490 observations. Finally, this dataset is further split, with 80% (9992) chunks used for actual model training and the remaining 20% (2498) reserved for validation during model training.

Then, the retrieval pipeline is implemented as described in detail in 3.2.

The only consideration to be made in this regard concerns the implementation of Approximate Nearest Neighbor (ANN) search. Databases like Neo4j, which to be more precise employs the Hierarchical Navigable Small World (HNSW) algorithm[89], implement ANN because, when dealing with large datasets, exact nearest neighbor searches can be computationally expensive and slow. ANN, on the other hand, significantly reduces query times by sacrificing some accuracy for speed. In the advanced configuration settings of the index two hyperparameters can be set to change the behavior of the algorithm. The first controls the maximum number of connections each node has in the HNSW graph. Increasing its value may lead to greater accuracy at the expense of increased index population and update times, especially for vectors with high dimensionality. The other one represents the number of nearest neighbors tracked during the insertion of vectors into the HNSW graph. Increasing this value improves the quality of the index, and may lead to greater accuracy (with diminishing returns) at the expense of increased index population and update times⁶. The choice of these parameters highly depends on each specific use-case and is left to the reader.

In the case of this thesis, numerous experiments have shown that the best

⁶neo4j.com/docs/cypher-manual/current/indexes/semantic-indexes/vector-indexes/#create-vector-index

performances (in terms of speed-accuracy balance) actually came from simply increasing the number of retrieved nodes. When interested in retrieving k nodes, the `numberOfNearestNeighbor` parameter of the query to the index is instead set to $k + n$, where n is an arbitrary parameter. The resulting $k + n$ nodes are then ordered by their score and only the top k nodes are actually extracted. This approach works because ANN algorithms divide the search space in “buckets”. By increasing the parameter to $k + n$, the index is asked to search a wider neighborhood of the vector space, i.e. the algorithm explores more buckets of the index, in this way considering a larger group of candidate vectors. This wider search can find points that the narrower search might miss due to the approximate nature of the algorithm.

All the results shown in this thesis are computed by retrieving a sample of 100 nodes, and then narrowing down to the top k matches.

4.3.1 Retrieval Metrics

To evaluate the quality of the retrieval pipeline we use three metrics:

1. **Precision** is defined as the ratio between the number of retrieved `Chunk` nodes that contain the answer and the total number of retrieved chunks. This metric quantifies the proportion of retrieved nodes that are actually relevant to answering the question. A low precision value implies that a significant portion of the context passed to the LLM for inference is irrelevant, potentially hindering the model’s ability to generate a correct response.
2. **Recall** is computed as the ratio between the number of retrieved `Chunk` nodes that contain the answer and the total number of chunks that contain the answer. This metric assesses how effectively the retrieval process captures the relevant information necessary for answering the query.
3. **Hit** is a binary metric that equals 1 if at least one of the retrieved nodes contains the answer and 0 otherwise. This measure is particularly useful for non-Multi-Hop questions, where the presence of a single relevant `Chunk` ensures that the answer is included in the context provided to the LLM, thereby enabling, though not ensuring, the RAG system to generate a correct response.
4. **Set Coverage** is a binary metric that equals 1 if all the necessary chunks required to answer the query are retrieved and 0 otherwise. This measure is crucial for Multi-Hop questions, where answering the query requires retrieving all the relevant chunks.

5. **F-Score** evaluates the trade-off between precision and recall. It is computed as

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}}$$

where β determines the relative importance of recall over precision. In the context of RAG systems, recall is often considered more critical than precision, as ensuring that the answer is present in the retrieved context is more important than filtering out irrelevant chunks. To reflect this, we report the F1-score (equal importance to precision and recall) alongside F2 and F3 scores, which progressively emphasize recall over precision.

One crucial consideration is that these metrics are significantly influenced by the number of chunks retrieved by the system. As the number of retrieved chunks increases, precision is likely to decrease, since the denominator of its ratio grows while the numerator may remain unchanged if all relevant chunks have already been retrieved. Conversely, recall tends to increase, as the denominator remains fixed while the probability of retrieving additional relevant chunks rises. The same holds for the Hit metric, which becomes more likely to be 1: the larger the retrieved sample, the higher the probability that at least one correct chunk is included.

The only metric that inherently addresses this trade-off is the F-score, as it balances precision and recall. However, determining the optimal balance remains a challenge, especially in the RAG domain, where these metrics do not necessarily carry the same weight. The choice of how many chunks to retrieve ultimately depends on the specific use case and is beyond the scope of this experiment.

To provide a comprehensive assessment, we report precision, recall, and hit rate for different retrieval sizes, and the decision of which dimensionality best suits one’s own specific case is left to the reader. In particular, *Accuracy-k*, *Recall-k*, *Hit-k* and *SetCoverage-k* measure the performance of the retrieval system when tasked with retrieving k chunks. In the evaluation on NQ-dev-20% we retrieve 1, 3, and 5 chunks, while for NQ-MH-100 we retrieve 4, 6, and 8 chunks. These values were chosen arbitrarily, with the intermediate value (3 and 6 respectively) set to be twice as large in the Multi-Hop evaluation, reflecting the need for retrieving at least twice the number of chunks in such cases.

4.3.2 Retrieval Performance

In this section, we compare the results of each retrieval approach. The results are calculated on the NQ-dev-20% test set and are presented in 4.1.

Retrieval Method	N. of Elements	N. of Chunks	Precision	Recall	Hit	F1	F2	F3
Vector Retrieval	1 Chunk	1	0.398	0.308	0.398	0.347	0.323	0.315
	3 Chunks	3	0.241	0.528	0.622	0.330	0.426	0.472
	5 Chunks	5	0.175	0.630	0.710	0.274	0.414	0.500
Page Retrieval	1 Page	~ 35	0.071	0.789	0.789	0.130	0.261	0.392
	2 Pages	~ 72	0.029	0.861	0.861	0.056	0.128	0.223
	3 Pages	~ 108	0.018	0.885	0.885	0.035	0.083	0.152
Section Retrieval	1 Section	~ 3	0.207	0.352	0.421	0.261	0.309	0.329
	3 Sections	~ 6	0.113	0.604	0.663	0.190	0.323	0.421
	5 Sections	~ 9	0.058	0.798	0.829	0.108	0.225	0.351
Entity Retrieval	1 Entity	~ 4	0.059	0.096	0.118	0.073	0.085	0.090
	3 Entities	~ 8	0.046	0.205	0.249	0.075	0.121	0.152
	5 Entities	~ 11	0.041	0.248	0.299	0.070	0.123	0.165
Path Retrieval	1 Path	~ 3	0.266	0.423	0.512	0.327	0.378	0.399
Naive Hybrid Retrieval	5 Chunks	5	0.188	0.695	0.776	0.296	0.451	0.547
Hybrid Retrieval	1 Chunk	1	0.478	0.371	0.478	0.418	0.388	0.379
	3 Chunks	3	0.298	0.662	0.756	0.411	0.532	0.590
	5 Chunks	5	0.215	0.780	0.851	0.337	0.511	0.618

Table 4.1: Retrieval Results on NQ-dev-20%

As anticipated in 4.2, **Vector Retrieval** serves as one of our main baselines. The findings indicate that retrieving five chunks includes the correct answer in 71% of cases. At the same time, the precision score remains reasonable, suggesting that, on average, approximately one out of every five retrieved chunks is relevant to the query. Although increasing the number of retrieved chunks could further boost the hit rate, excessively long contexts, such as those containing ten or more concatenated chunks, might negatively

impact the LLM’s response quality, especially if the retrieved chunks are presented in an improper order. In contrast, retrieving three chunks offers a more conservative yet still effective balance between recall and precision. However, when retrieving only one chunk, the performance significantly drops, with the correct node appearing in just 40% of the cases, making it a less viable option for reliable retrieval.

Page Retrieval is a broad and expansive approach, as it retrieves all chunks associated with n Wikipedia pages. Given this wide retrieval scope, it is unsurprising that the method yields low precision scores. However, its high recall rate suggests that it is effective in narrowing down the search space, making it a valuable first step for the Path Retrieval module. Notably, the relevant chunks are found within the top three retrieved pages 89% of the time.

Section Retrieval appears to yield better results, particularly when extracting five entities, where it achieves a more balanced trade-off between hit rate and precision. While the hit rate is slightly lower than that of Page Retrieval with three pages extracted, the precision is three times higher. However, despite these improvements, its performance still falls significantly short of Vector Retrieval. When comparing precision at the same level (one parent extracted for Section Retrieval and three chunks extracted for Vector Retrieval) the hit rate of Vector Retrieval is substantially higher, outperforming Section Retrieval by 20 percentage points.

Entity Retrieval appears to be the weakest method among those evaluated so far. With only one entity extracted, its precision is half that of Section Retrieval when extracting three parent entities, and its hit rate is merely a third.

The results of the **Path Retrieval** are among the most promising. Although its hit rate is 10 percentage points lower than that of Vector Retrieval with three extracted chunks, it achieves the highest precision among all models. This indicates that the model is highly effective in identifying the correct chunks while minimizing the inclusion of irrelevant paragraphs. By reducing the presence of non-relevant content, it helps maintain a cleaner context, ultimately preventing confusion for the LLM and preserving the quality of its responses.

The **Naive Hybrid Retrieval**, presented as one of our main baselines in section 4.2, is the only method cited in this section that is not incorporated in the hybrid retrieval, but that serves solely as a point of comparison with existing works presented in the state of the art (section 2.7.4). Compared to vector retrieval with the same number of retrieved chunks (5), Naive Hybrid Retrieval demonstrates improved precision and recall, aligning with the findings of the HybridRAG paper. While the gain in precision is marginal,

the improvement in recall is more significant, increasing from 63% to nearly 70%. This leads to a hit rate that is approximately 7 percentage points higher, indicating that, on average, Naive Hybrid Retrieval retrieves at least one correct answer 7% more often than vector retrieval. These results highlight the benefits of incorporating a graph-based retrieval component, even in a simple concatenation-based hybrid approach.

The results unequivocally demonstrate the superiority of our **Hybrid Retrieval** approach over the baseline Vector Retriever and all the other individual methods across all evaluated metrics. Even when retrieving just a single chunk, our model consistently outperforms the baseline, achieving significant improvements in precision, recall, and detection. As we increase the number of retrieved chunks, the gap between the two methods becomes even more pronounced. With three retrieved chunks, Hybrid Retrieval substantially enhances recall and detection, capturing a broader spectrum of relevant information while maintaining a slight advantage in precision. Notably, with just three chunks, Hybrid Retrieval successfully retrieves at least one relevant chunk in 76% of cases, meaning that in three out of four instances, the model receives the necessary information to generate a correct response. In contrast, Vector Retrieval achieves this only 62%. This trend persists with five retrieved chunks, where the hit rate reaches an impressive 85%, a 14 percentage-point improvement over Vector Retrieval and 7 over the Naive Hybrid Retrieval. All this is done while maintaining a strong precision of 21.5%, ensuring that the model is not provided with useless context that could degrade the quality of its response.

The strongest confirmation of the advantage of our Hybrid method over any other retrieval approach comes from the F-scores, which we have identified as the most important metrics due to their ability to balance precision and recall. Across all three reported measures (F1, F2, and F3), which differ in the weight assigned to recall relative to precision, our method consistently achieves the highest scores compared to all other retrieval systems, and does so with a large margin. This demonstrates that, regardless of the user’s preferred trade-off between the relevance ratio of the extracted context (precision) and the likelihood of retrieving the necessary chunks (recall), our approach delivers superior performance.

N. of Chunks	Retrieval Method	Metrics					
		Precision	Recall	Set Coverage	F1	F2	F3
4	Vector	0.175	0.357	0.090	0.235	0.296	0.323
	Hybrid Naive	0.125	0.381	0.110	0.188	0.270	0.316
	Hybrid	0.302	0.599	0.330	0.402	0.501	0.545
6	Vector	0.138	0.418	0.160	0.207	0.297	0.347
	Hybrid Naive	0.143	0.432	0.170	0.215	0.308	0.359
	Hybrid	0.232	0.688	0.430	0.347	0.494	0.575
8	Vector	0.111	0.448	0.180	0.178	0.279	0.344
	Hybrid Naive	0.155	0.467	0.200	0.232	0.333	0.389
	Hybrid	0.180	0.711	0.480	0.287	0.447	0.549

Table 4.2: Retrieval Results on NQ-MH-100

The performance gap between our hybrid retrieval method and the others becomes even more pronounced when evaluated on the Multi-Hop dataset, as highlighted by the results presented in Table 4.2. With 4 retrieved chunks, the probability of retrieving all relevant chunks for a given query is three times higher with our method than both the other two approaches. At 6 retrieved chunks, the vector retrieval correctly retrieves all relevant chunks in 16% of cases. The Naive Hybrid Retrieval method performs slightly better than pure vector retrieval, demonstrating the benefits of incorporating a graph structure and leveraging our graph traversal mechanism for Multi-Hop question answering. However, its simplistic approach of merely concatenating results from two retrieval methods significantly limits the graph’s full potential. This is evident in the hit rate, where Naive Hybrid Retrieval reaches only 17%, compared to the 43% achieved by our approach. Even as k increases to 8, hybrid retrieval continues to maintain a substantial lead, with a Recall of 71% compared to 45% for vector retrieval and 47% of Naive Hybrid Retrieval.

Even in the results for multi-hop questions, the significant improvement of our system over the two baseline methods is strongly supported by the F-score results. Regardless of the chosen trade-off between precision and recall, as represented by different beta values (F1, F2, F3), our approach consistently achieves the highest scores across all retrieval settings.

Furthermore, it is important to emphasize once again that the results

of our Hybrid Retrieval on the Multi-Hop dataset are obtained *without any retraining*, meaning that we use the same neural network trained on the NQ-dev-80% training dataset. This demonstrates the model’s ability to generalize effectively to Multi-Hop questions, despite never having been explicitly trained on them. The fact that the model performs well in this setting highlights the robustness of our retrieval approach and its capacity to handle more complex reasoning tasks without requiring additional fine-tuning.

4.4 Evaluation of the RAG System

In this section, we integrate both retrieval techniques into a complete Question Answering pipeline using RAG. This allows us to assess whether the superior retrieval quality demonstrated by our hybrid system effectively translates into improved answer accuracy, finally answering to our last research question.

Since evaluating answer quality requires a ground truth, this assessment cannot be applied to the entire NQ-dev-20% test set. The primary limitation comes from the NQ dataset itself, where some questions are only linked to the chunk containing the answer (“long-answer” annotation) without explicitly specifying the exact answer itself (“short answer” annotation). To ensure a reliable evaluation, we restrict our analysis to questions for which at least one annotator has provided a short answer, using these as reference points to assess the accuracy of our RAG-generated responses. We define this subset of the NQ-dev-20% test set, composed only of questions with a short-answer annotation, as NQ-dev-20%-sa. This refined test set serves as the benchmark for evaluating the performance of our RAG system. However, given that our main focus is on retrieval quality, this constraint does not significantly impact our overall evaluation. In contrast, for the Multi-Hop dataset, since it was synthetically constructed, we have complete access to full answer annotations.

For each question in the NQ-dev-20%-sa test set, we retrieve 3 chunks using Vector Retrieval and 3 chunks using our Hybrid Retrieval. The retrieved context is then fed into an LLM (GPT-4o) along with the original question. We use the prompt illustrated in 4.4 and parse the generated responses.

```
[caption={System Prompt for Answer Generation}]
Imagine being an extractive Question Answering System.
Respond to the user's question extracting the smallest possible slice of
text from the context, without changing anything.
If the answers are multiple, separate the extracted slices of text with
a forward slash ("/").
If no answer to the user's query can be found in the context, output "'
ND'.
In any case, do not add any comments to your answer.
```

For the Multi-Hop test set, we retrieve 6 chunks instead of 3, compensating for the fact that Multi-Hop questions require additional information to be answered correctly. The same methodology is applied, but with a carefully designed prompt (showed in 4.4) ensuring that the LLM extracts answers strictly from the provided context rather than relying on its parametric knowledge.

```
[caption={System Prompt for MultiHop Answer Generation}]
Imagine being a question answering system.
You are not interested in giving always the right answer, but in
evaluating the informative content of the context, and therefore
answering the question using ONLY the information that can be
extracted from the context.
You are not interested in assessing whether the statements contained in
the context are true, but in reasoning over the context and answer
the question based on its statements.
If the question CANNOT be answered from the context alone, output the
tag 'ND'.
Else, if the question can be answered using ONLY the informations
present in the text, output the answer that can be inferred using
only the information present in the context.
```

4.4.1 RAG Metrics

To evaluate the quality of the RAG’s responses, instead, we employ four different metrics:

1. **Exact match** is the most naive metric, simply comparing if the generated answer is exactly the same as the ground truth.
2. **ROUGE-L** measures the overlap between the generated response and the reference text based on their longest common subsequence.
3. **Cosine Similarity** measures the cosine similarity between the vector representations of the generated response and the ground truth, evaluating how similar the two are in terms of their semantic content.
4. **RAGAS: Factual Correctness** evaluates the extent to which the generated response aligns with the reference. The metric, developed by RAGAS⁷[23], employs an LLM to decompose both the response and the reference into individual claims. Subsequently, it utilizes natural language inference techniques to assess the factual overlap, quantified using a F1 score, between these claims.

⁷docs.ragas.io

4.4.2 RAG Performance

		Retrieval	EM	ROUGE	Cosine	FC
NQ-dev-20%-sa	Vector	0.355	0.461	0.891	0.636	
	Hybrid	0.399	0.515	0.903	0.672	
NQ-MH-100	Vector	0.040	0.394	0.843	0.424	
	Hybrid	0.060	0.507	0.887	0.519	

Table 4.3: Comparison of Vector and Hybrid RAG methods in MultiHop and Non-MultiHop QA.

The evaluation results of the RAG-generated answers confirm that the superior retrieval effectiveness of our hybrid system translates into a higher accuracy of the final responses. Table 4.3 shows that across all metrics, ranging from simpler ones to more complex measures, our pipeline consistently outperforms vector retrieval with statistically significant improvements.

Similar to the retrieval results, where the performance gap widened in the Multi-Hop dataset, the advantages of the hybrid approach become even more pronounced in the RAG evaluation for Multi-Hop QA. Notably, the cosine similarity metric is, on average, 5% higher with the hybrid method, while the factual correctness score shows an even greater improvement, increasing from 4% in the standard dataset to 9% in the Multi-Hop dataset. These results highlights the hybrid method’s ability to retrieve and utilize more relevant information in complex reasoning tasks.

4.5 Additional Evaluations and Findings

In this section, we present additional results obtained during the research to provide a broader perspective on the effectiveness of our proposed system. Section 4.5.1 examines the importance of each feature in identifying the most relevant chunk. This analysis allows us to assess which retrieval methods play a crucial role in locating the chunk containing the correct answer and which have a lesser impact. Section 4.5.2 demonstrates the effectiveness of the methodologies applied during the experiment, as described in section 3.2: local standardization (3.2.2) and active sampling (3.2.2). Finally, Section 4.5.3 presents the model’s performance when trained on a significantly

reduced dataset. This evaluation helps determine whether our pipeline remains applicable in scenarios where only a limited amount of training data is available.

4.5.1 Features Importance in Retrieval

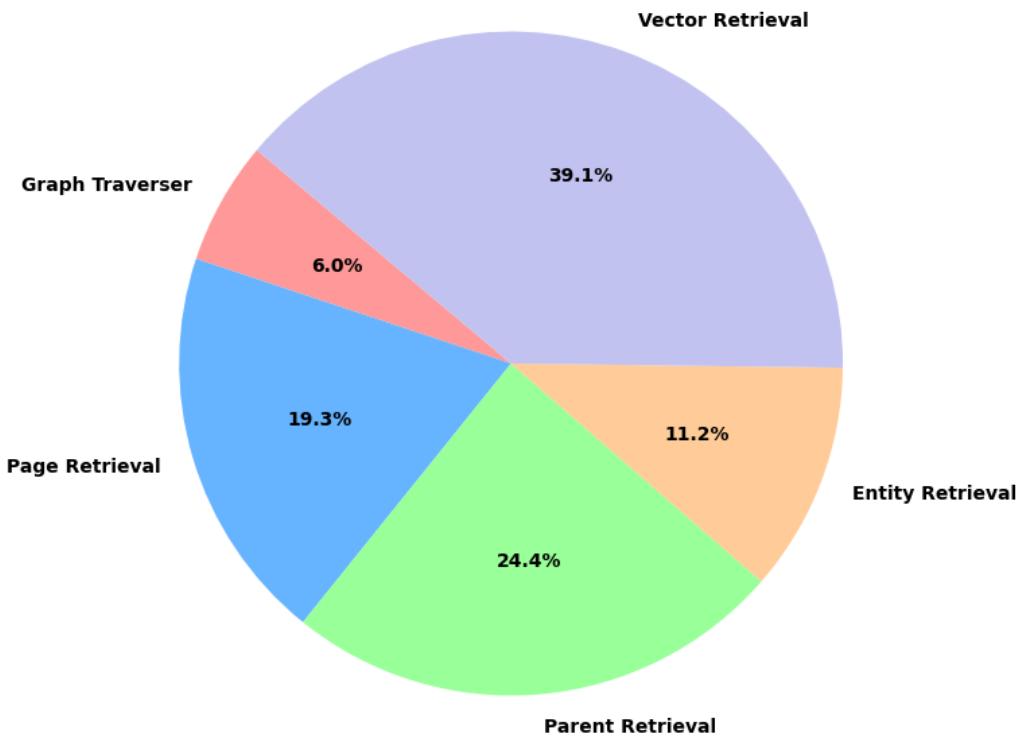


Figure 4.2: Features Importance in Retrieval

The subgoal of the second objective was: “Understanding which retrieval methods play a more crucial role in identifying the most relevant passages, and subsequently designing a retrieval approach that weights different retrieval methods proportionally to their actual importance”. While we developed a retrieval method that assigns weights to its components based on their importance in identifying relevant passages, a key question remains: what are the actual importances of these retrieval methods?

To address this, we trained a Random Forest model on the same NQ-dev-80% training data extracted from the NQ dataset’s development set.

Interpreting the results of a FFNN is inherently complex, making it difficult to directly assess feature importance. Although the Random Forest and FFNN are not identical models, the Random Forest achieves comparable results, albeit slightly lower than the FFNN. This suggests that the feature importance scores derived from the Random Forest, which are much easier to extract, can serve as a reasonable proxy for their importance in the neural network. The results of this analysis are presented in Figure 4.2.

Unsurprisingly, the most relevant retrieval method is Vector Retrieval, which accounts for 39% of the importance in identifying the correct answer. This confirms that, despite other contributing factors, the content of each chunk remains the most critical feature. However, it is noteworthy that this method alone represents less than 40% of the final selection process. This finding reinforces the idea, already highlighted by the retrieval and RAG results, that retrieval methods relying exclusively on semantic similarity fail to capture a significant portion of relevant information.

The second most important methods, Section Retrieval and Page Retrieval, are those directly related to the structure and organization of the document. These account for 24.4% and 19.3% of the importance respectively. Interestingly, their combined importance (43.7%) surpasses that of Vector Retrieval alone, suggesting that a chunk's relevance is determined not only by its content but also by its structural position within a relevant document and chapter, with both factors contributing approximately equally.

The least influential methods are Entity Retrieval (11.2%) and Path Retrieval (6%). Nevertheless, together they contribute 17.1% to the retrieval process, demonstrating that entity-based methods, which leverage semantic relationships within the document, play a meaningful role.

It is particularly interesting to observe that Path Retrieval, despite being the most complex retrieval method, has the lowest marginal importance (6%). This suggests that the effectiveness of the retrieval process does not stem from the complexity of a method like Path Retrieval, which relies on an LLM, but rather from the intelligent and non-trivial combination of relatively simple methods. When Path Retrieval is removed from the model's training, the retrieval performance on the test set (for the top 3 extracted chunks) drops significantly, with Precision: 28%, Recall: 61%, and Hit Rate: 71%. These results are considerably lower than those obtained with Path Retrieval included, demonstrating its utility for achieving maximum effectiveness. However, even without Path Retrieval, the performance remains notably better than Vector Retrieval alone. The corresponding F1, F2, and F3 scores without Path Retrieval are 0.384, 0.494, and 0.546, respectively. This means that our approach still outperforms the Naive Hybrid Retrieval baseline in F1 and F2 and achieves comparable results in F3. These findings

confirm that the key factor behind our method’s success is not the use of more complex retrieval techniques but rather the well-designed integration of multiple retrieval strategies.

4.5.2 Evaluation of Key Methodological Choices

Method	Precision	Recall	Hit
Vector Retrieval	0.241	0.528	0.622
Global Standardization	0.271	0.602	0.693
Local Standardization	0.292	0.648	0.745
Active Sampling	0.298	0.662	0.756

Table 4.4: Effectiveness of the solutions implemented (3 chunks)

This section is dedicated to demonstrating the effectiveness of all the methodological choices presented in the Methodology section. The table 4.4 shows a progression starting from vector retrieval, with each step incorporating an additional methodology: Global Standardization, which is the method used to train our model without applying the local standardization described in Section 3; Local Standardization, implemented without the active sampling described in Section 4; and finally, the results of our final hybrid model. Each result is evaluated on the NQ-dev-20% test set. As shown in the table, each methodological refinement leads to a performance increase over all the evaluated metrics, demonstrating that every step in our approach has contributed to enhancing the model’s effectiveness.

4.5.3 Performance with Limited Training Data

In this section, we shift our focus to the evaluation of the performance of our hybrid model when trained on a limited dataset. To achieve this, we select an even smaller subset from the original dev set of the NQ dataset. Our subset consisted of a sample of 200 observations, balanced to include 100 instances where the independent variable is `_answer` equals 1, and 100 instances where it equals 0. We then retrain the model using this limited dataset. The resulting performance metrics are detailed in table 4.5.

N. of Chunks	Retrieval Method	Metrics		
		Precision	Recall	Hit
1	Vector	0.398	0.308	0.398
	Hybrid ST	0.391	0.299	0.391
	Hybrid	0.478	0.371	0.478
3	Vector	0.241	0.528	0.622
	Hybrid ST	0.256	0.560	0.657
	Hybrid	0.298	0.662	0.756
5	Vector	0.175	0.630	0.710
	Naive Hybrid	0.188	0.695	0.776
	Hybrid ST	0.190	0.683	0.766
	Hybrid	0.215	0.780	0.851

Table 4.5: Performance with Limited Training Data

As the results indicate, retraining the model with a significantly reduced training dataset led to a corresponding decrease in performance. However, the performance remained reasonably good, still surpassing the vector retrieval method. Specifically, the hybrid retrieval with the reduced training sample achieved a hit rate equal to the vector retrieval when extracting one chunk. Moreover, when extracting three chunks, the hybrid retrieval showed a hit rate increase of almost four percentage points. Finally, the hit rate improved of five percentage points when extracting five chunks. Despite these improvements, the hybrid retrieval trained with a small dataset is still substantially inferior to the hybrid retrieval trained with the complete dataset and performs on par, at least when extracting five chunks, with the naive retrieval.

This demonstrates that those who wish to develop a pipeline similar to ours, but have a significantly smaller dataset, could still obtain advantageous results from implementing our pipeline instead of simply using vector retrieval. Nevertheless, as previously shown in the evaluation of our pipeline on the Multi-Hop dataset, retraining the model is not always necessary.

Chapter 5

Conclusion and Future Developments

The objective of this thesis was to design a complete retrieval approach, ranging from document structuring to the retrieval model itself, that could determine the most relevant chunks by integrating and combining all the informational dimensions of the original documents. These informational dimensions encompass three key aspects, either explicitly or implicitly embedded in the document: its content, its structure, and the entities mentioned within it, along with the semantic relationships that interconnect them. We argue that a retrieval system incapable of fully leveraging all these dimensions is insufficient to effectively handle all query types.

Our approach was developed in two main phases. First, we aimed to construct a document representation that explicitly preserves and organizes all informational dimensions. Second, we sought to develop a retrieval system capable of utilizing this enriched structure to maximize retrieval effectiveness. For this second objective, we aimed at designing a hybrid approach that could merge relevance scores from different retrieval methods. Each individual method would assign a relevance score to a chunk based on its respective informational dimension, and our model would then integrate these scores into a single comprehensive relevance measure. Crucially, we aimed to make this integration not trivial: since some retrieval methods are more effective than others in identifying the correct chunks, our model must consequently weight their contributions rather than treating them as equally important.

In Section 2, we reviewed the state-of-the-art research in document structuring and hybrid retrieval. We found that no existing approach explicitly represents all informational dimensions of a document, inevitably leading to a loss of potentially valuable information during retrieval. As a consequence, since retrieval performance is inherently constrained by the representation of

the underlying KB, we did not identify any system capable of fully leveraging all dimensions to determine the most relevant chunks. While most retrieval methods focus on a single dimension, some hybrid approaches that incorporate multiple dimensions do exist. However, these methods do not combine individual retrieval scores into a unified ranking, but they either merge the sets of retrieved chunks from each method or apply retrieval techniques sequentially, filtering chunks at each stage.

In Section 3, we outlined our approach to achieving the first two main objectives. To address the first objective, Section 3.1 introduced a KG-based document representation designed to explicitly preserve all informational dimensions. In this representation, “Entity” nodes are created to represent the entities mentioned within documents and are interconnected based on relations extracted from DBpedia. Note that, since we work with encyclopedic documents, Entity and Pages are the same thing, since each entity is represented by its corresponding Wikipedia page. The hierarchical structure of the document is captured through “SubChapter” nodes, which maintain links to their corresponding “Entity” nodes. These structural elements enhance content navigation by guiding the retrieval algorithm toward the most relevant document sections. Finally, the textual content is segmented into smaller “Chunk” nodes, each linked to its parent Entity or SubChapter, as well as to adjacent chunks and referenced entities.

To achieve the second objective, we trained a Neural Network on a subset of the development set from Google’s NQ dataset (NQ-dev-80%) to integrate retrieval scores from multiple independent methods. Specifically, three retrieval approaches, “Section Retrieval”, “Page Retrieval”, and “SubChapter Selector”, were employed to estimate a chunk’s relevance based on its position within the document structure, effectively leveraging the document organization as captured by our knowledge representation. An entity-centric perspective was introduced through the “SubGraph Retrieval” and “Entity Retrieval” methods, which exploit entities and their relationships. The SubChapter Selector and SubGraph Retrieval were then combined into a “Path Retrieval” mechanism, designed to traverse the graph and identify the most relevant path for a given query. Finally, a “Vector Retrieval” method is tasked with retrieving the most relevant chunk purely based on its textual content. The Neural Network is trained to integrate these diverse sources, ultimately determining which Chunk nodes to retrieve to answer the user’s query. In this way, each retrieval method contributes with a distinct perspective on a chunk’s relevance, and the network synthesizes these insights to generate the final response.

To evaluate our approach, in section 4.1 we introduce two testing datasets. First, we assess our hybrid retrieval system on the remaining portion of the

development set from Google’s NQ dataset (NQ-dev-20%), comparing its performance against each individual retrieval method and a new baseline. This baseline mimics hybrid state-of-the-art approaches by naively combining chunks extracted through a Graph Traverser with those retrieved via Vector Retrieval. Additionally, we evaluate our method on a synthetic MultiHop dataset consisting of 100 questions generated from the same KB.

The results in Section 4.3.2 unequivocally demonstrate the superiority of our Hybrid Retrieval approach over both the baseline Vector Retriever and all individual retrieval methods across all evaluated metrics. The strongest evidence supporting the effectiveness of our method comes from the F scores, which we identify as the most critical metrics due to their ability to balance precision and recall. Across all three reported measures (F1, F2, and F3), which vary in relative weight assigned to recall and precision, our approach consistently achieves the highest scores, outperforming all other retrieval systems by a significant margin. This demonstrates that, regardless of the preferred trade-off between the relevance ratio of the retrieved context (precision) and the likelihood of retrieving all necessary chunks (recall), our method delivers superior performance. Furthermore, in Section 4.4.2, we show that this improved retrieval precision translates directly into higher-quality responses generated by the RAG system.

The results on the Multi-Hop dataset follow the same trend, demonstrating an even larger performance gap compared to other retrieval methods and, consequently, a higher quality of the responses generated by the RAG system. This outcome is particularly remarkable given that the Neural Network is not retrained for Multi-Hop queries, but the model originally trained to integrate retrieval scores on NQ-dev-80% is directly applied to retrieve relevant chunks for these entirely new Multi-Hop questions, highlighting a remarkable ability of the model to generalize to previously unseen types of questions.

Finally, in Section 4.5, we present additional findings obtained during our investigation to provide a broader perspective on the effectiveness of our proposed system. In particular, Section 4.5.1 analyzes the relative contribution of each retrieval method. One of the most noteworthy insights is that Path Retrieval, despite being the most complex retrieval approach (since it leverages an LLM) exhibits the lowest marginal importance. This suggests that the overall effectiveness of our retrieval process does not stem from the complexity of a single method like Path Retrieval, but rather from the strength of our approach: intelligently and non-trivially combining multiple relatively simple retrieval methods.

In addition, in this section, we validate key methodological choices and demonstrate that our approach remains promising, albeit less impressive, even when trained on a very limited dataset.

Despite the strong results achieved, certain limitations must be addressed in future research to fully realize the potential of our method. One notable limitation stems from its ability to be trained on a small dataset. While this characteristic is encouraging, the resulting performance remains on par with a Naive Hybrid Approach. However, the evaluation of the MultiHop dataset suggests that our model does not always require retraining to effectively handle new types of questions. Future research should further investigate this generalization capability by evaluating the model on entirely new datasets without retraining. If we can confirm that a model trained on our dataset consistently generalizes across different tasks, then the observed performance limitations on small training sets would become far less significant.

One of the most significant limitations is that, in this thesis, we have used Wikipedia pages as the initial unstructured data collection on which we applied our method. This choice allowed us to leverage Wikipedia hyperlinks for parsing entity mentions and to extract relationships using parallel databases like DBpedia. This ensured high-quality extractions and simplified the process, allowing us to focus on the main objectives of this thesis: structuring the extracted information and applying a hybrid retrieval system on this structure. However, in the more general case of truly unstructured documents, where techniques like NER, NEL, coreference resolution, and relation extraction would need to be applied to extract such information, our approach's performance would be heavily dependent on the accuracy of these extraction methods. As the performance of these methods degrades, so too would the performance of our approach. Therefore, it is necessary for future research to evaluate the impact of extracting information entirely from scratch using these models.

Another limitation lies in the weighting mechanism of our retrieval methods. Although our approach adjusts the importance of each retrieval method based on its effectiveness in identifying the most relevant chunk, it does not dynamically adapt to the specific query being asked. Some queries may be more structured, thus benefiting more from hierarchical retrieval, while others may be less structured and better served by entity-based retrieval. Future research might explore adaptive weighting mechanisms.

An interesting direction for future research could involve the fine-tuning of the parameters selected during the development process. Parameters such as the number of nodes extracted for each type of retrieval, the default cosine similarity assigned to missing values (0.75), the percentage of chunks retained among the hard negatives during active sampling (10%), and the initial nodes for the SubGraph Retriever (3) were chosen based on heuristic reasoning rather than rigorous scientific justification. Consequently, there is potential for future studies to identify more optimal combinations of these

parameters, which could further enhance the overall performance and quality of the model. This exploration could lead to a more robust and efficient framework, offering significant improvements in the retrieval process and the resulting graph-based data analysis.

Scalability presents another key challenge. While our knowledge graph representation effectively integrates multiple informational dimensions, its computational cost can escalate significantly when applied to large-scale document collections. As highlighted in our experimental section, parsing 5,223 Wikipedia pages into our graph representation took 36 minutes. Although this duration is manageable for most real-world applications, it could become a bottleneck in open-domain question answering scenarios, where our approach might be required to parse the entirety of Wikipedia. Our approach is not inherently slow; rather, optimization was not our primary focus as we concentrated on validating the method itself. Future research could focus on enhancing scalability for larger datasets, for instance, by incorporating parallelization in the parsing process.

Lastly, integrating multiple retrieval methods introduces complexity into the system. While this fusion improves effectiveness, it may also reduce interpretability, making it challenging to understand the relative contribution of each method and diagnose potential errors. By incorporating methods to clarify the contribution of each retrieval component and providing insights into why a particular chunk was selected, users can build trust in the system, and developers can better diagnose and refine the retrieval process.

References

- [1] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.
- [2] Gemini Team et al. “Gemini: a family of highly capable multimodal models”. In: *arXiv preprint arXiv:2312.11805* (2023).
- [3] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.
- [4] Yunfan Gao et al. “Retrieval-augmented generation for large language models: A survey”. In: *arXiv preprint arXiv:2312.10997* (2023).
- [5] Zhuyun Dai et al. “Promptagator: Few-shot dense retrieval from 8 examples”. In: *arXiv preprint arXiv:2209.11755* (2022).
- [6] Peitian Zhang et al. “Retrieve anything to augment large language models”. In: *arXiv preprint arXiv:2310.07554* (2023).
- [7] Zichun Yu et al. “Augmentation-adapted retriever improves generalization of language models as generic plug-in”. In: *arXiv preprint arXiv:2305.17331* (2023).
- [8] Xintao Wang et al. “Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases”. In: *arXiv preprint arXiv:2308.11761* (2023).
- [9] Xiaoxin He et al. “G-retriever: Retrieval-augmented generation for textual graph understanding and question answering”. In: *arXiv preprint arXiv:2402.07630* (2024).
- [10] Minki Kang et al. “Knowledge graph-augmented language models for knowledge-grounded dialogue generation”. In: *arXiv preprint arXiv:2305.18846* (2023).

- [11] Riccardo Pozzi et al. “Combining Knowledge Graphs and NLP to Analyze Instant Messaging Data in Criminal Investigations”. In: *International Conference on Web Information Systems Engineering*. Springer. 2025, pp. 427–442.
- [12] Cyril Zakka et al. “Almanac—retrieval-augmented language models for clinical medicine”. In: *NEJM AI* 1.2 (2024), A1oa2300068.
- [13] Zhentao Xu et al. “Retrieval-augmented generation with knowledge graphs for customer service question answering”. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2024, pp. 2905–2909.
- [14] Forbes Technology Council. “The Big Unstructured Data Problem”. In: *Forbes* (2017). URL: <https://www.forbes.com/sites/forbestechcouncil/2017/06/05/the-big-unstructured-data-problem/>.
- [15] Tam Harbert. “Tapping the power of unstructured data”. In: *MIT Sloan. Feb* 1 (2021), p. 3.
- [16] Yikun Han, Chunjiang Liu, and Pengfei Wang. “A comprehensive survey on vector database: Storage and retrieval technique, challenge”. In: *arXiv preprint arXiv:2310.11703* (2023).
- [17] Jing Li et al. “A survey on deep learning for named entity recognition”. In: *IEEE transactions on knowledge and data engineering* 34.1 (2020), pp. 50–70.
- [18] Lingfeng Zhong et al. “A comprehensive survey on automatic knowledge graph construction”. In: *ACM Computing Surveys* 56.4 (2023), pp. 1–62.
- [19] Qianqian Zhang, Mengdong Chen, and Lianzhong Liu. “A review on entity relation extraction”. In: *2017 second international conference on mechanical, control and computer engineering (ICMCCE)*. IEEE. 2017, pp. 178–183.
- [20] Xibin Dong et al. “A survey on ensemble learning”. In: *Frontiers of Computer Science* 14 (2020), pp. 241–258.
- [21] Omer Sagi and Lior Rokach. “Ensemble learning: A survey”. In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 8.4 (2018), e1249.
- [22] Tom Kwiatkowski et al. “Natural questions: a benchmark for question answering research”. In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 453–466.

- [23] Shahul Es et al. “Ragas: Automated evaluation of retrieval augmented generation”. In: *arXiv preprint arXiv:2309.15217* (2023).
- [24] Lynette Hirschman and Robert Gaizauskas. “Natural language question answering: the view from here”. In: *natural language engineering* 7.4 (2001), pp. 275–300.
- [25] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. “The question answering systems: A survey”. In: *International Journal of Research and Reviews in Information Sciences (IJRRIS)* 2.3 (2012).
- [26] Eric Brill, Susan Dumais, and Michele Banko. “An analysis of the AskMSR question-answering system”. In: *Proceedings of the 2002 conference on empirical methods in natural language processing (EMNLP 2002)*. 2002, pp. 257–264.
- [27] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of naacl-HLT*. Vol. 1. Minneapolis, Minnesota. 2019, p. 2.
- [28] P Rajpurkar. “Squad: 100,000+ questions for machine comprehension of text”. In: *arXiv preprint arXiv:1606.05250* (2016).
- [29] A Vaswani. “Attention is all you need”. In: *Advances in Neural Information Processing Systems* (2017).
- [30] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.
- [31] Kevin Meng et al. “Locating and editing factual associations in GPT”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 17359–17372.
- [32] Fabio Petroni et al. “Language models as knowledge bases?” In: *arXiv preprint arXiv:1909.01066* (2019).
- [33] Adam Roberts, Colin Raffel, and Noam Shazeer. “How much knowledge can you pack into the parameters of a language model?” In: *arXiv preprint arXiv:2002.08910* (2020).
- [34] Gary Marcus. “The next decade in AI: four steps towards robust artificial intelligence”. In: *arXiv preprint arXiv:2002.06177* (2020).
- [35] Jason Weston, Sumit Chopra, and Antoine Bordes. “Memory networks”. In: *arXiv preprint arXiv:1410.3916* (2014).

- [36] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. “End-to-end memory networks”. In: *Advances in neural information processing systems* 28 (2015).
- [37] Armand Joulin and Tomas Mikolov. “Inferring algorithmic patterns with stack-augmented recurrent nets”. In: *Advances in neural information processing systems* 28 (2015).
- [38] Guillaume Lample et al. “Large memory layers with product keys”. In: *Advances in Neural Information Processing Systems* 32 (2019).
- [39] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. “Latent retrieval for weakly supervised open domain question answering”. In: *arXiv preprint arXiv:1906.00300* (2019).
- [40] Kelvin Guu et al. “Retrieval augmented language model pre-training”. In: *International conference on machine learning*. PMLR. 2020, pp. 3929–3938.
- [41] Vladimir Karpukhin et al. “Dense passage retrieval for open-domain question answering”. In: *arXiv preprint arXiv:2004.04906* (2020).
- [42] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-scale similarity search with GPUs”. In: *IEEE Transactions on Big Data* 7.3 (2019), pp. 535–547.
- [43] M Lewis. “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *arXiv preprint arXiv:1910.13461* (2019).
- [44] OpenAI. *New and improved embedding model*. <https://openai.com/index/new-and-improved-embedding-model/>. 2022.
- [45] Xianming Li and Jing Li. “Angle-optimized text embeddings”. In: *arXiv preprint arXiv:2309.12871* (2023).
- [46] VoyageAI. *Voyage’s embedding models*. <https://docs.voyageai.com/docs/embeddings>. 2023.
- [47] BAAI. *Flagembedding*. <https://github.com/FlagOpen/FlagEmbedding>. 2023.
- [48] Luyu Gao et al. “Precise zero-shot dense retrieval without relevance labels”. In: *arXiv preprint arXiv:2212.10496* (2022).
- [49] Ron Litman et al. “Scatter: selective context attentional scene text recognizer”. In: *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11962–11972.

- [50] Nathan Anderson, Caleb Wilson, and Stephen D Richardson. “Lingua: Addressing scenarios for live interpretation and automatic dubbing”. In: *Proceedings of the 15th Biennial Conference of the Association for Machine Translation in the Americas (Volume 2: Users and Providers Track and Government Track)*. 2022, pp. 202–209.
- [51] Nelson F Liu et al. “Lost in the middle: How language models use long contexts”. In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.
- [52] Amit Singhal et al. “Introducing the knowledge graph: things, not strings”. In: *Official google blog* 5.16 (2012), p. 3.
- [53] Aidan Hogan et al. “Knowledge graphs”. In: *ACM Computing Surveys (Csur)* 54.4 (2021), pp. 1–37.
- [54] Shirui Pan et al. “Unifying large language models and knowledge graphs: A roadmap”. In: *IEEE Transactions on Knowledge and Data Engineering* (2024).
- [55] Zhengyan Zhang et al. “ERNIE: Enhanced language representation with informative entities”. In: *arXiv preprint arXiv:1905.07129* (2019).
- [56] Corby Rosset et al. “Knowledge-aware language model pretraining”. In: *arXiv preprint arXiv:2007.00655* (2020).
- [57] Bill Yuchen Lin et al. “Kagnet: Knowledge-aware graph networks for commonsense reasoning”. In: *arXiv preprint arXiv:1909.02151* (2019).
- [58] Zhiyuan Zhang et al. “Pretrain-KGE: learning knowledge representation from pretrained language models”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. 2020, pp. 259–266.
- [59] Abhijeet Kumar et al. “Building knowledge graph using pre-trained language model for learning entity-aware relationships”. In: *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE. 2020, pp. 310–315.
- [60] Jian Guan, Yansen Wang, and Minlie Huang. “Story ending generation with incremental encoding and commonsense knowledge”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 6473–6480.
- [61] Hao Zhou et al. “Commonsense knowledge aware conversation generation with graph attention.” In: *IJCAI*. 2018, pp. 4623–4629.
- [62] Bin He et al. “Integrating graph contextualized knowledge into pre-trained language models”. In: *arXiv preprint arXiv:1912.00147* (2019).

- [63] Xiaozhi Wang et al. “KEPLER: A unified model for knowledge embedding and pre-trained language representation”. In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 176–194.
- [64] Xiaoyan Wang et al. “Improving natural language inference using external knowledge in the science questions domain”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 7208–7215.
- [65] Chao Feng, Xinyu Zhang, and Zichu Fei. “Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs”. In: *arXiv preprint arXiv:2309.03118* (2023).
- [66] Sören Auer et al. “Dbpedia: A nucleus for a web of open data”. In: *international semantic web conference*. Springer. 2007, pp. 722–735.
- [67] Kurt Bollacker, Robert Cook, and Patrick Tufts. “Freebase: A shared database of structured general human knowledge”. In: *AAAI*. Vol. 7. 2007, pp. 1962–1963.
- [68] Denny Vrandečić and Markus Krötzsch. “Wikidata: a free collaborative knowledgebase”. In: *Communications of the ACM* 57.10 (2014), pp. 78–85.
- [69] George A Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* 38.11 (1995), pp. 39–41.
- [70] Muhammad Mahbubur Rahman. *Understanding the logical and semantic structure of large documents*. University of Maryland, Baltimore County, 2018.
- [71] André V Duarte et al. “Lumberchunker: Long-form narrative document segmentation”. In: *arXiv preprint arXiv:2406.17526* (2024).
- [72] Weijie Chen et al. “KG-Retriever: Efficient Knowledge Indexing for Retrieval-Augmented Large Language Models”. In: *arXiv preprint arXiv:2412.05547* (2024).
- [73] Darren Edge et al. “From local to global: A graph rag approach to query-focused summarization”. In: *arXiv preprint arXiv:2404.16130* (2024).
- [74] Ye Liu et al. “Dense hierarchical retrieval for open-domain question answering”. In: *arXiv preprint arXiv:2110.15439* (2021).
- [75] Yu Wang et al. “Knowledge graph prompting for multi-document question answering”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 17. 2024, pp. 19206–19214.

- [76] Qiang Sun et al. “Docs2KG: Unified Knowledge Graph Construction from Heterogeneous Documents Assisted by Large Language Models”. In: *arXiv preprint arXiv:2406.02962* (2024).
- [77] Zhilin Yang et al. “HotpotQA: A dataset for diverse, explainable multi-hop question answering”. In: *arXiv preprint arXiv:1809.09600* (2018).
- [78] Paolo Ferragina and Ugo Scaiella. “Fast and accurate annotation of short texts with wikipedia pages”. In: *IEEE software* 29.1 (2011), pp. 70–75.
- [79] Xingxuan Li et al. “Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources”. In: *arXiv preprint arXiv:2305.13269* (2023).
- [80] Yuyu Zhang et al. “Variational reasoning for question answering with knowledge graph”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.
- [81] Jing Zhang et al. “Subgraph retrieval enhanced model for multi-hop knowledge base question answering”. In: *arXiv preprint arXiv:2202.13296* (2022).
- [82] Jiaxin Shi et al. “Transfernet: An effective and transparent framework for multi-hop question answering over relation graph”. In: *arXiv preprint arXiv:2104.07302* (2021).
- [83] Bhaskarjit Sarmah et al. “Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction”. In: *Proceedings of the 5th ACM International Conference on AI in Finance*. 2024, pp. 608–616.
- [84] Nguyen Nam Doan et al. “A Hybrid Retrieval Approach for Advancing Retrieval-Augmented Generation Systems”. In: *Proceedings of the 7th International Conference on Natural Language and Speech Processing (ICNLSP 2024)*. 2024, pp. 397–409.
- [85] 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)*. IEEE. 2024, pp. 166–169.
- [86] Wenhui Chen et al. “Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks”. In: *arXiv preprint arXiv:2211.12588* (2022).

- [87] Daniel Bienstock et al. “A note on the prize collecting traveling salesman problem”. In: *Mathematical programming* 59.1 (1993), pp. 413–420.
- [88] Xin Wang, Yudong Chen, and Wenwu Zhu. “A survey on curriculum learning”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.9 (2021), pp. 4555–4576.
- [89] Yu A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.
- [90] Barlas Oguz et al. “Unik-qa: Unified representations of structured and unstructured knowledge for open-domain question answering”. In: *arXiv preprint arXiv:2012.14610* (2020).
- [91] Peter Buneman et al. “Adding structure to unstructured data”. In: *Database Theory—ICDT’97: 6th International Conference Delphi, Greece, January 8–10, 1997 Proceedings* 6. Springer. 1997, pp. 336–350.

Acronyms

- AKGC** Automatic Knowledge Graph Construction. 22
- ANN** Approximate Nearest Neighbor. 68, 69
- DPR** Dense Passage Retrieval. 16, 17
- FFNN** Feed Forward Neural Network. 55, 58, 79
- IR** Information Retrieval. 12, 13, 30
- KB** Knowledge Base. 2, 10, 17–19, 26, 27, 29, 31–33, 35, 37, 39, 59, 60, 83, 84
- KG** Knowledge Graph. 4–8, 11, 19–23, 25, 27, 28, 31, 33–38, 41, 44, 48, 54, 59, 63, 64, 83
- KGQA** Knowledge Graph Question Answering. 29
- LLM** Large Language Model. 1, 2, 10–16, 18–27, 30–32, 34–38, 43, 51–54, 64, 66, 69, 72, 75, 76, 79, 84
- NEL** Named Entity Linking. 6, 37, 60, 85
- NER** Named Entity Recognition. 6, 13, 22, 31, 35, 37, 60, 85
- NLP** Natural Language Processing. 13–15, 17, 22
- NQ** Natural Question. 10, 16, 18, 59–62, 64, 75, 78, 80
- OWL** Web Ontology Language. 20
- PCST** Price Collecting Steiner Tree. 29, 34
- POS** Part-of-Speech. 13

QA Question Answering. 1–3, 11–18, 23, 29, 36, 60, 64, 67, 77, 98

RAG Retrieval Augmented Generation. 2–4, 10, 12–14, 17–19, 22, 23, 25, 29–32, 34, 36, 38, 44, 48, 64, 68, 69, 75–77, 79

RDF Resource Description Framework. 20, 22

Seq2Seq Sequence-to-Sequence. 2, 17

TF-IDF Term Frequency-Inverse Document Frequency. 27, 31

VRN Variational Reasoning Network. 33

List of Figures

1.1	Comparison of different document representations	5
1.2	Documents structuring into the Knowledge Base	6
1.3	Comparison of Retrieval Systems leveraging informational dimensions individually	7
1.4	Retrieval Pipeline	9
2.1	RAG architecture	17
3.1	Complete Approach	40
3.2	Example of the proposed solution for structuring a collection in the KG	41
3.3	Graph Structure	42
3.4	Example of the proposed solution for chunks retrieval	46
3.5	Path Retrieval	51
3.6	Hybrid Retrieval	54
4.1	Wikipedia Parser	61
4.2	Features Importance in Retrieval	78

List of Tables

2.1	State-of-the-Art Methods for Document Structuring	24
2.2	Comparison of State-of-the-Art Methods for RAG retrieval . .	29
3.1	Summary table of the retrieval methods employed.	48
4.1	Retrieval Results on NQ-dev-20%	71
4.2	Retrieval Results on NQ-MH-100	74
4.3	Comparison of Vector and Hybrid RAG methods in MultiHop and Non-MultiHop QA.	77
4.4	Effectiveness of the solutions implemented (3 chunks)	80
4.5	Performance with Limited Training Data	81