ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Department of Computer Science and Engineering
Computer Science and Engineering

UniBot: A Retrieval-Augmented Conversational Agent for the University of Bologna

Bachelor Thesis in
Data Intensive Applications

Supervisor
Prof. Gianluca Moro

Candidate
Nicolas Amadori

Co-supervisors
Dr. Lorenzo Molfetta
Dr. Giacomo Frisoni

Third Session
Academic Year 2023 – 2024

KEY WORDS

Large Language Models

Web Scraping

Data Extraction

Generative Chatbot

Retrieval-Augmented Generation

Ai miei genitori

Abstract

Recently, the utility and convenience of Large Language Models (LLMs), especially as generative chatbots, have gained significant attention in various contexts. Chatbots are designed with simplicity and intuitiveness, leveraging natural conversational interfaces that align with user habits. They can effectively handle a wide range of tasks, from everyday inquiries to complex professional applications. However, the current limitations of these chatbots often restrict their functionality, primarily in responding to general public information queries. This constraint diminishes their effectiveness, particularly in specialized fields that demand more granular and contextualized information. This thesis proposes developing a prototype of a conversational agent specifically designed to meet the needs of the University of Bologna. The primary objective is to enable the chatbot to respond comprehensively to all types of questions a student might ask related to the university context. This goal will be achieved by integrating authentic information ensuring the chatbot can respond precisely and accurately to student inquiries. To facilitate this, the project will implement a Retrieval-Augmented Generation pipeline that utilizes a customized University of Bologna-specific information dataset, scraped and cleaned from the official university website. This innovative approach combines LLMs' advanced natural language understanding and response generation capabilities with targeted academic knowledge. Preliminary results indicate a significant enhancement in the chatbot's performance, with an improvement of approximately 25-30% in its ability to respond to questions relevant to the university setting, compared to direct generation using the same language model. This allows the chatbot to provide detailed answers on topics like course structures and Erasmus programs.

Introduction

Currently, all information that a university student may need is accessible through the official web resources of the University of Bologna; however, this information is often distributed across various pages and documents, making it challenging to locate specific data and, more generally, leading to information dispersion.

Our thesis project aims to provide students and members of the university community with a tool that simplifies information retrieval and answers their inquiries. Essential features of this tool include intuitive usability and ease of interaction—characteristics inherent to generative chatbots powered by Large Language Models. Such chatbots facilitate natural language interaction through a familiar interface in the form of a chat, making the experience straightforward and user-friendly.

Students may ask the same questions they would typically direct to the university's administrative office, but with the advantage of 24/7 availability and instantaneous responses. These inquiries may range from general questions about specific courses to more precise requests, such as application deadlines or payment schedules for tuition fees. Therefore, our model must understand the context of the question, retrieve relevant information, and provide the most accurate answer possible.

The work detailed in the following text begins with the creation of a custom dataset by obtaining and cleaning information from the university's web resources. This information is then used to populate the vector database utilized for retrieval.

A crucial component for evaluating the results involved collecting Frequently Asked Questions (FAQs) and their corresponding answers from university

resources. This allowed for the establishment of a question set to test the model's generated responses against authentic answers. Subsequently, creating a new dataset featuring multiple-choice questions became necessary to enable a more refined assessment of the model's performance.

Next, the configuration and use of Large Language Models are presented to enable them to perform the various necessary tasks, such as responding to identified FAQs, generating multiple-choice questions, and creating alternative responses for the original FAQs.

Building on the contextual groundwork we have established, the upcoming chapters of this thesis offer a comprehensive examination of our research, clarifying our methodologies and showcasing our results and are organized as follows:

- **Chapter 1 - Theoretical Framework**: This chapter will elucidate the fundamental concepts underlying the thesis project. It will begin with an in-depth exploration of the functioning of large language models and their application in creating a chatbot. Subsequently, the structure of the Retrieval-Augmented Generation (RAG) processes will be articulated, focusing on response generation utilizing contextual documents.

- **Chapter 2 - Related Work**: In this chapter, I will enumerate the significant scientific works shaping our research. These studies have inspired us and provided a context for our work.

- **Chapter 3 - Method**: This chapter provides an in-depth examination of our proposed methodology, beginning with collecting and transforming source documents. It will continue with configuring the Large Language Models (LLMs) and culminate in establishing the RAG process to achieve the best possible results.

- **Chapter 4 - Experimental Setup**: In this chapter, we describe the experimental setup utilized during our study. We discuss the employed libraries, the metrics adopted for performance evaluation, implementation specifics, and the hyperparameters involved. This thorough overview of

our experimental setup lays the groundwork for the following chapter, in which we will share our results.

- **Chapter 5 - Results and Discussion**: This chapter outlines the outcomes of our experiments and tests. We analyze the data, highlighting the findings demonstrating the proposed approach's effectiveness. Through extensive discussions, we offer meaningful interpretations of these results, fostering a deeper comprehension of the implications and contributions of our research.

- **Conclusion and Future Work**: This concluding section summarizes our thesis, integrating our research's main findings, implications, and contributions. We contemplate the importance of our work and its possible impact. Furthermore, we outline potential directions for future research, pinpointing unexplored areas and opportunities to expand upon the groundwork we have laid.

Index

List of Figures

List of Tables

Chapter 1

Theoretical Framework

This chapter provides an overview and explanation of the core concepts essential to this thesis project, focusing on artificial intelligence (AI), machine learning, natural language processing (NLP) and Retrieval-Augmented Generation.

1.1 Generative AI

Generative artificial intelligence (AI) is a transformative technology that leverages complex algorithms, such as neural networks, to autonomously produce new content, ranging from text and images to music and programming code. Figure 1.1 illustrates the hierarchical relationship among Artificial Intelligence, Machine Learning, Generative AI, and LLMs through a representation of nested sets. Unlike traditional AI, which primarily focuses on recognizing patterns or making predictions based on existing data, generative AI creates new outputs by learning the underlying structure of a dataset and generating similar yet unique content. This capability is primarily driven by transformer-based architectures, such as GPT (Generative Pre-trained Transformer) models. These models are trained on vast datasets and use probabilistic techniques to predict and assemble coherent outputs, resulting in remarkably human-like and contextually relevant creations. Generative AI is already widely applied across various industries, from automating creative tasks in media and design to creating chatbots, like

in our case, granting everybody access to such an incredible tool.

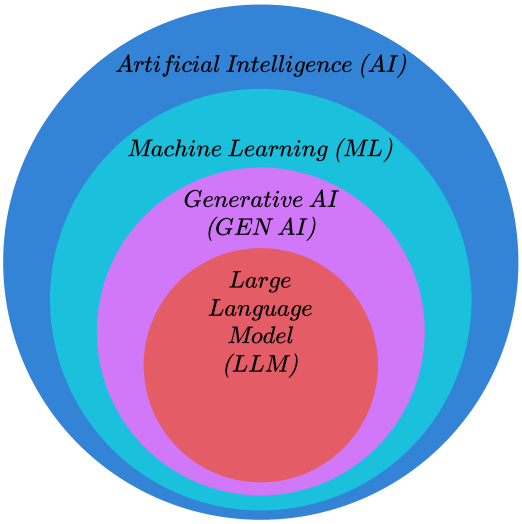

Figure 1.1: Relationship between AI, ML, Generative AI, and LLMs. AI is the overarching field of intelligent systems, ML is a subset focused on learning from data, Generative AI creates new content, and LLMs are specialized Generative AI models for language understanding and generation.

1.2 Large Language Models

Large Language Models are advanced neural networks built on the transformer architecture, which enables training these models on extensive datasets of textual information. This architecture has facilitated the scalability necessary for processing vast amounts of data. When researchers began to significantly increase the size of these models and expose them to large datasets, they demonstrated remarkable capabilities, such as comprehending complex and nuanced language and generating text with unprecedented eloquence. Engineers have effectively trained these models by leveraging extensive datasets to recognize the statistical patterns inherent in natural language. As a result, LLMs excel in understanding and producing human language.

Initially, machine learning models were trained for specific tasks, such as sentiment analysis or language translation, each requiring unique model training and adaptation. However, this approach is now largely outdated, as LLMs can execute many tasks by simply modifying the input prompt without requiring model retraining or adaptation.

LLMs' versatility distinguishes them from earlier models. A single model can seamlessly handle tasks such as chat interactions, copywriting, translation, text summarization, brainstorming, code generation, and question answering, among others. This multifunctionality not only showcases the transformative potential of LLMs across various domains but also emphasizes the importance of prompt engineering, as crafting well-designed prompts directly impacts the performance and accuracy of LLM outputs across diverse tasks.

1.3 Prompt Engineering

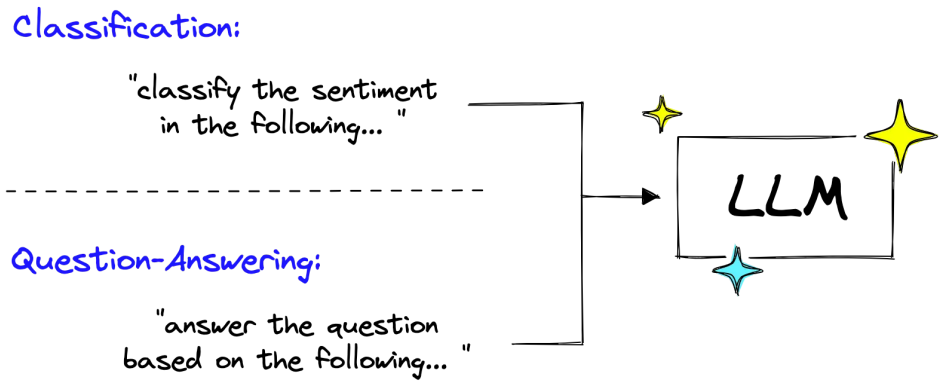

Figure 1.2: Various prompt examples for LLMs. Using different prompts allows you to adjust the task that the LLM performs. Source: `www.pinecone.io`.

Prompt engineering has become essential for boosting pre-trained LLMs functionality. This technique involves crafting specific task-oriented prompts or instructions to direct the model's output without changing its parameters. The importance of prompt engineering is evident in its profound effect on the flexibility of LLMs. By refining model responses through well-designed prompts, prompt engineering allows these models to perform effectively across a wide

range of tasks and fields (visualized in Figure 1.2). This flexibility contrasts with traditional approaches, which usually require model retraining or intensive fine-tuning to achieve task-specific performance [5].

In prompt engineering, there are three different types of prompting: zero-shot, one-shot, and few-shot:

- **Zero-shot Prompting:** The model receives only a prompt without examples, relying solely on its general training to understand and respond to the task.
- **One-shot Prompting:** The prompt includes a single example of the desired output before the actual task.
- **Few-shot Prompting:** The model is provided with multiple examples within the prompt, illustrating the pattern or format of the expected output.

Each approach serves different needs, balancing efficiency with accuracy, and supports a wide range of applications in LLMs.

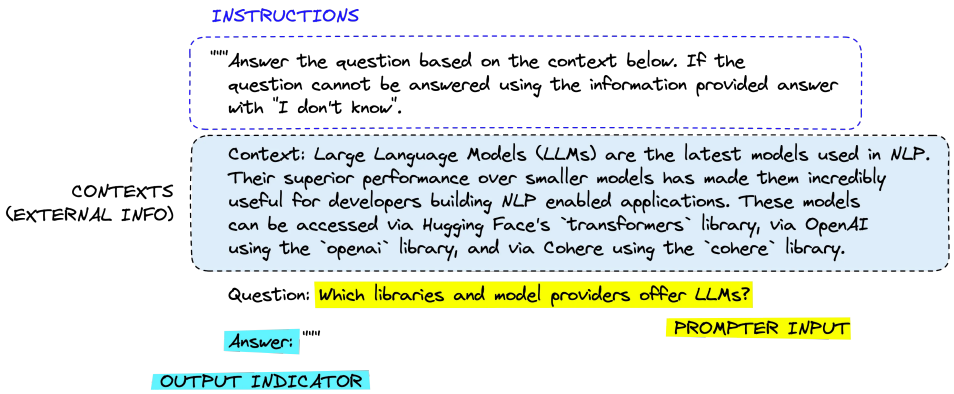

Figure 1.3: Example of an LLM prompt template: Differentiating between roles helps the LLM better interpret the instructions by separating general behavior guidelines from the specific task instruction. Source: `www.pinecone.io`.

While standard language models are robust, they may not consistently produce responses that reflect user intentions, particularly when prompts lack clarity or when tasks necessitate precise direction. Numerous models are

optimized to utilize specific prompts with roles that distinguish instructions from the underlying data. For most models three distinct roles are expected for the LLMs prompt template: system, user, and assistant. The system role sets the parameters and behavior of the LLM, outlining how it should respond and the tone to adopt. The user role represents the individual seeking information, driving the conversation through questions and requests. Finally, the LLM fulfils the assistant role, which generates responses based on the system's guidelines. Together, these roles create a framework for effective communication, with the system providing structure, the user initiating dialogue, and the assistant delivering informative responses (visible in Figure 1.3).

1.4 Retrieval-Augmented Generation

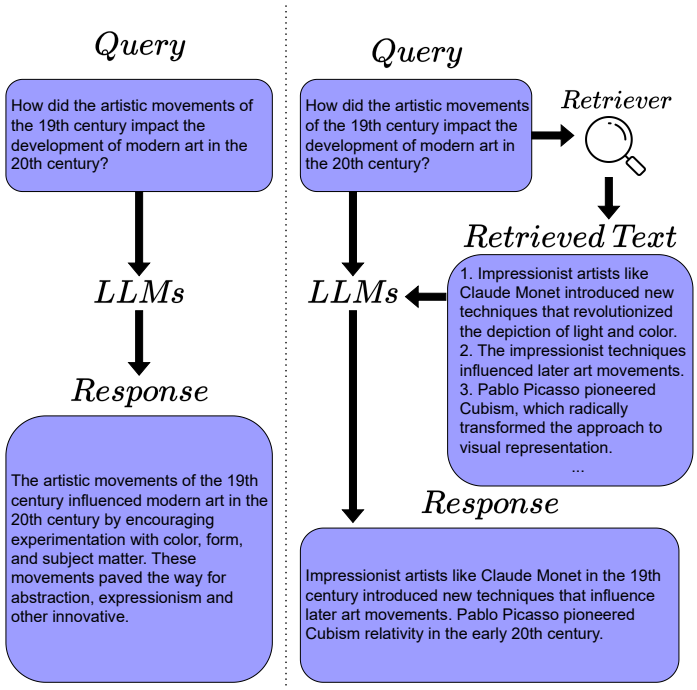

Figure 1.4: Comparison between Direct LLM and RAG: The RAG workflow includes a retrieval step between the query and the LLM's response generation phase.

Retrieval-Augmented Generation is one of the most advanced techniques in AI, offering reliable and current external knowledge that significantly simplifies many tasks. In the AI-generated content (AIGC) age, RAG's robust retrieval capabilities allow it to provide supplementary knowledge, enhancing generative AI's ability to produce high-quality outputs. LLMs have recently shown groundbreaking skills in understanding and generating language. However, they still have certain limitations, such as generating inaccurate information (hallucinations) and relying on outdated internal data. RAG's strength in delivering up-to-date, relevant information has led to the development of Retrieval-Augmented Large Language Models [6]. These models draw on external, authoritative knowledge bases rather than relying solely on their internal data, significantly improving LLMs' output quality. As shown in Figure 1.4, the key distinction between the direct LLM approach and RAG is that RAG includes an additional retrieval phase. In this phase, relevant documents and information are gathered and combined with the original query before being processed by the LLM.

As shown in Figure 1.5, RAG can be split into two main phases: Retrieval and Generation.

- **Retrieval:** The phase where relevant documents and information are gathered to enhance the LLM response context.

- **Generation Augmentation:** The phase where input query and retrieved pieces of information are combined to make the LLM generate the best answer for the question.

1.4.1 Retrieval

In the context of RAG, efficiently retrieving relevant documents from the data source is crucial. Several key issues are involved, such as the retrieval type, retrieval granularity, indexing and query optimization, and selection of the embedding model.

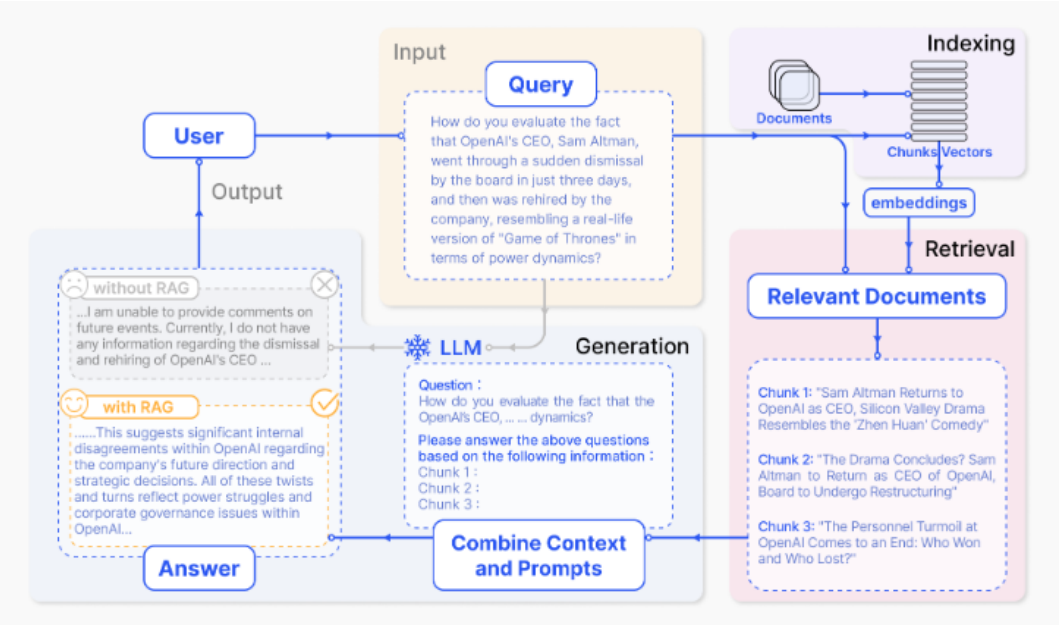

Figure 1.5: A representative instance of the RAG process applied to question answering. Every RAG phase is shown: input phase, indexing phase, retrieval phase and generation phase. Source: "Retrieval-Augmented Generation for Large Language Models" paper [1].

Retriever Type

Retrieval methods are divided into two main types: sparse and dense. Sparse retrieval methods are word-based and mainly applied for text retrieval. They are commonly applied in RAG to enrich input for models or supply examples for in-context learning, though their fixed term-based nature limits adaptability. Dense retrieval encodes queries and documents into continuous vector spaces, usually with trainable models, often BERT-based. Dense retrieval generally surpasses sparse methods when fine-tuned for specific tasks, showing greater flexibility and effectiveness, especially in open-domain QA and varied retrieval tasks [6].

Retriever Granularity

Retrieval granularity refers to the unit at which a corpus is indexed, such as document, passage, token, or entity level. In RAG, the choice of granularity significantly affects model performance, impacting both effectiveness and efficiency by influencing database storage requirements and the computational cost of searches [7].

Indexing Optimization

During the Indexing phase, documents will undergo processing, segmentation, and transformation into embeddings for storage within a vector database. The quality of the index construction is critical, as it directly influences the retrieval phase's ability to obtain the appropriate context. In this stage, implementing an effective chunking strategy is essential to divide the documents into segments of a predetermined token count, optimized for the dataset, while incorporating a specific degree of text overlap. Additionally, attaching metadata to these chunks can facilitate the filtering of retrieved documents, thereby narrowing the scope of the retrieval process. Finally, establishing a structural index for the documents, utilizing a hierarchical framework, can enhance the efficiency of data retrieval and processing relevant information [1].

Query optimization

One of the primary challenges associated with RAG is its direct dependence on the user's original query as the foundation for information retrieval. Users do not always articulate precise or clear questions; moreover, the questions they formulate may be complex or poorly structured. To address these issues, techniques utilizing LLMs can be employed, such as query expansion—expanding the original question to capture a broader context—and query transformation—retrieving information based on a modified version of the query rather than solely relying on the user's initial formulation [1].

Embedding

Text embeddings are vector representations of natural language that encapsulate semantic information (a visual representation in Figure 1.6. These embeddings are extensively utilized across various NLP tasks, including information retrieval (IR), question answering, semantic textual similarity, bitext mining, item recommendation, and more. In RAG systems, the retrieval process is fundamentally based on calculating the similarity between the embeddings of user queries and document chunks. This similarity is often measured using techniques such as cosine similarity, which quantifies the cosine of the angle between two vectors in a high-dimensional space. The semantic representation

Figure 1.6: A visualization of word embeddings, where words are transformed into a high-dimensional vector representation that captures their meanings and relationships with other words in a continuous space. Source: `www.datastax.com`.

capabilities of the embedding models used heavily influence the effectiveness of this retrieval process. Embeddings serve as dense vector representations of questions and document chunks, capturing their underlying semantic meanings.

High-quality embeddings facilitate the identification of relevant documents even when there is no direct lexical overlap between the query and the content. This is particularly important in natural language processing tasks, where the nuances of language, synonyms, and contextual meaning must be considered [8].

1.4.2 Generation Augmentation

Augmentation describes the technical process that integrates retrieval and generation parts, which is essential in RAG systems. Three different integrations can be used to obtain the best results in relation to the LLM used and the dataset characteristics: input-layer, output-layer and intermediate-layer [6].

- **Input-Layer Integration:** This is the most common way to integrate generation and the pieces of information retrieved. It consists of combining them and jointly passing them to the generator, resulting in a straightforward to-implement approach that, despite the effectiveness, can be limited due to the fact that the model must process the context altogether with the question and can result in inefficient when the retrieved context is too wide.

- **Output-Layer Integration:** This integration combines the retrieved context and the generated answer; the retrieval result is used to refine and correct the generated information. This approach grants much more control over the final quality of the answer but can be more challenging to implement.

- **Intermediate-Layer Integration:** This is the more complex integration type since it needs to design a semi-parametric module to integrate the retrieved results through the internal layers of the generation model. This approach can be more efficient because specific information can be retrieved during the generation, but it also requests substantial modifications to the model.

1.5 Graph Retrieval-Augmented Generation

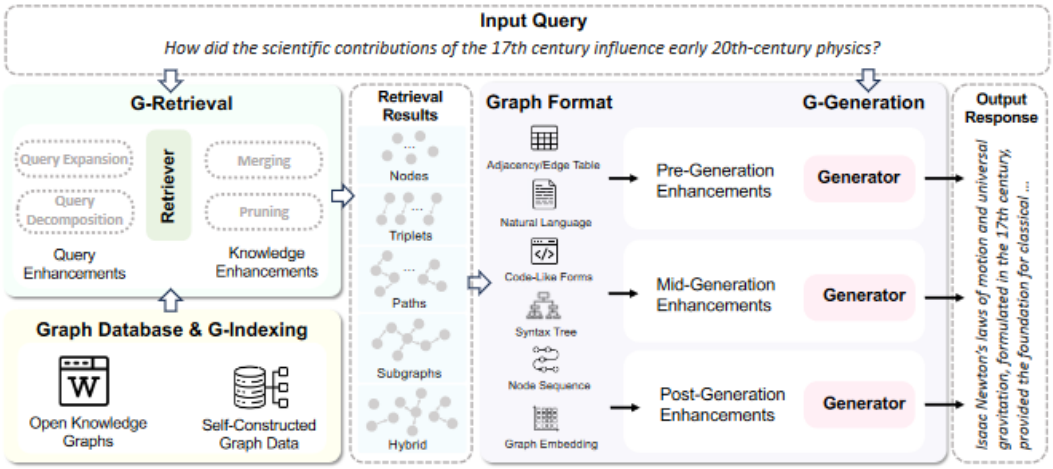

Figure 1.7: A visualization of the GraphRAG process, highlighting the similarities between the naive RAG and GraphRAG implementations. While both processes share comparable phases, GraphRAG distinguishes itself by incorporating knowledge graphs. Source: "Graph Retrieval-Augmented Generation" paper [2]

Graph Retrieval-Augmented Generation (GraphRAG) is a framework that leverages external structured knowledge graphs (an example in Figure 1.8) to improve contextual understanding of LMs and generate more informed responses. GraphRAG technique is an evolution of RAG, with the difference in the utilization of the knowledge base to retrieve from; in fact, instead of retrieving simple text chunks from a vector database identified by a similarity search a knowledge graph is generated and used to identify relationships between entities, giving contextual information to the llm for him to generate a perfect response. The overall workflow of GraphRAG, as illustrated in Figure 1.7, comprises these stages that we are going to discuss: Graph-Based Indexing, Graph-Guided Retrieval and Graph-Enhanced Generation [2].

1.5.1 Graph-Based Indexing

This is the initial phase of GraphRAG, known as Graph-Based Indexing; the objective is to establish or select an appropriate graph database tailored to support subsequent tasks. This graph database may be sourced from publicly available knowledge graphs, pre-existing graph datasets, or constructed from proprietary data such as textual information. The indexing process entails defining node and edge properties, linking interconnected nodes, and organizing the data structure to facilitate efficient traversal and retrieval. Often, LLMs are used to create these graphs, leveraging their contextual understanding. This phase plays a crucial role in enhancing query performance and retrieval speed by determining the granularity of retrieval.

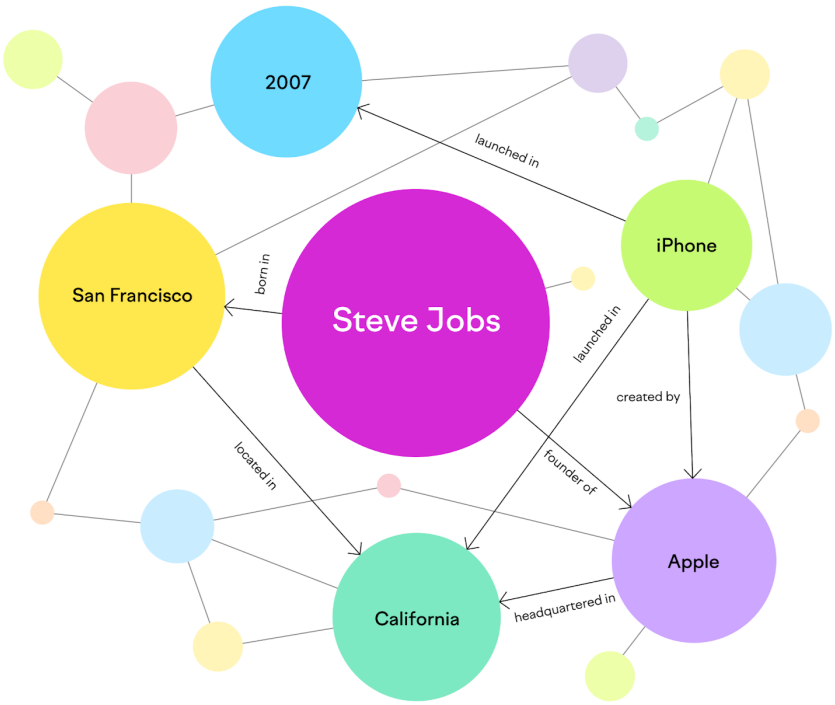

Figure 1.8: An example of knowledge graph. As shown, nodes represent entities, while edges represent the relationships between them. Source: `www.semrush.com`.

1.5.2 Graph-Guided Retrieval

Subsequent to the graph-based indexing phase, the graph-guided retrieval phase is dedicated to extracting relevant information from the graph database in response to user input. In fact, given a user query articulated in natural language, the retrieval stage aims to identify and extract the most pertinent components, such as entities, triples, paths, and subgraphs within the knowledge graph. The type of information retrieved depends on the specific implementation of the GraphRAG workflow and may vary based on the nature of the underlying knowledge. This process ensures the returned information aligns closely with the user's informational needs.

1.5.3 Graph-Enhanced Generation

The final phase involves generating a response to the initial user query by leveraging the contextual information obtained during retrieval. In this stage, the retrieved graph data is transformed into formats compatible with response generators. These generators utilize both the user query and the processed graph data as inputs to produce a comprehensive and accurate response. Integrating relevant knowledge during response generation ensures that the language model has access to all essential information, thereby optimizing the quality and relevance of the generated response. Even in this phase, various implementations are available and can be chosen

1.6 Website Crawling for RAG systems

Collecting webpages systematically is a sophisticated task that requires automation to maintain an up-to-date information dataset for the Retrieval Dataset. A web crawler, also known as a spider, is a highly effective tool for this purpose. It is a software application or programmed script designed to browse the World Wide Web systematically and automatically.

The growing demand for advanced information retrieval systems has increased the popularity of website crawling, especially with the rise of RAG systems. Unlike traditional language models that rely solely on pre-trained

data, RAG systems incorporate real-time information from external sources, resulting in more accurate and relevant responses.

A key advantage of RAG systems is their ability to use continuously updated information without frequent retraining. Businesses can deploy dedicated RAG solutions with web crawlers to gather the latest industry data, ensuring their language models remain current and enhancing decision-making and customer interactions.

As organizations recognize the importance of real-time data retrieval, the role of web crawlers will become increasingly critical, highlighting the need for advanced crawling technologies in modern data-driven strategies.

1.6.1 Crawling Spiders

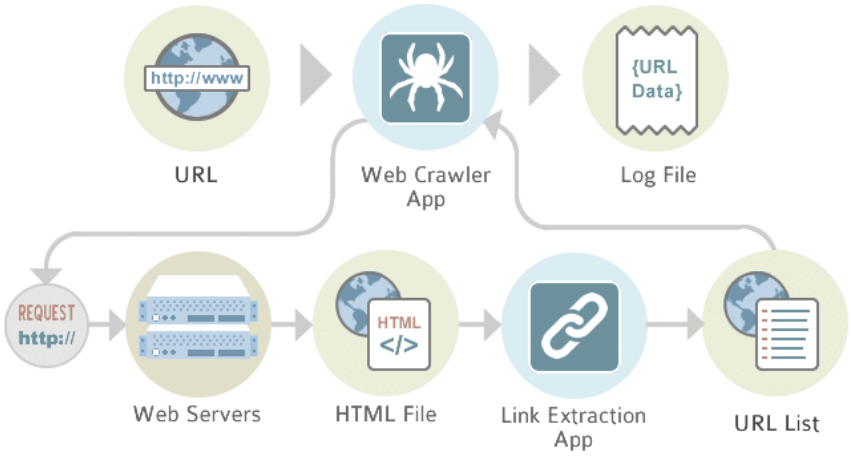

Figure 1.9: A visualization of the website crawling process, emphasizing the recursive structure of the crawler. Notably, for each URL visited, additional URLs are collected for subsequent visits, enabling continuous exploration of the website. Source: `www.researchgate.net`.

The World Wide Web is structured as a directed graph, where each webpage serves as a node and each hyperlink as an edge. In this structure, locating and gathering information can be conceptualized as traversing a directed graph. By following hyperlinks embedded within webpages, a web crawler is able to explore and retrieve content from multiple interconnected pages, starting from

a single page and expanding to newly discovered links (as shown in Figure 1.9. This way, the crawler leverages the Web's graph-like architecture to automate comprehensive data collection across interconnected resources [9].

Three website crawling techniques can be used while scraping websites based on the goal you want to obtain.

- **General purpose crawling:** A general purpose crawler collects as many pages as possible from a set of URLs and their links. General-purpose crawling can slow down the bandwidth since it fetches all the pages. The crawling could take days or weeks without any limitation regarding the domain or the depth of the link tree.

- **Focused crawling:** A focused crawler is designed to collect documents only on a specific topic, which reduces the amount of network traffic, and its purpose is to gather a set of pages which match some characteristics selectively.

- **Distributed crawling:** In a distributed crawler, multiple processes are used to parallelize the scraping.

1.6.2 Web Data cleaning

A critical component in web data crawling is preparing and structuring data provided to LLMs. Adequate data preparation for LLMs requires careful consideration of factors such as data organization and clarity. Although LLMs are proficient in processing natural language due to extensive training, they often encounter challenges when handling HTML-formatted data, including substantial non-informative elements that confuse the model and degrade input quality.

To address these issues, tools for converting HTML content into LLM-friendly text have gained popularity and sophistication in recent years. Some of these tools rely on regular expressions (regex) to transform HTML; however, recent advancements have introduced small language models specifically designed to enhance the efficiency and accuracy of data conversion and cleaning tasks.

Chapter 2

Related Work

2.1 Web Data Cleaning Tools

As mentioned before, tools to prepare web data for LLMs are becoming increasingly requested since the rising popularity of rag systems.

One of the first company which offered tools for the described task is Jina AI, a leading search AI company, that at the moment is proposing 2 of them: Reader API (a solution using regex and similar) and Reader Language Models (a solution using a small language model) [10].

Before deep diving into the differences between the two tools offered by Jina AI, Figure 2.1 shows the different pipelines obtained by using them.

2.1.1 Reader API by Jina AI

Reader API facilitates extracting key information from web pages by isolating and converting the primary content into a streamlined text format suitable for LLMs. By filtering out extraneous elements, the Reader API ensures that only the essential content is retained, thereby generating high-quality input for RAG systems and, more in general, improving the overall performance of LLMs in applications that rely on real-time, web-based information. This process optimizes the input data by eliminating irrelevant or redundant information and enhances the efficiency of downstream tasks, as LLMs can engage with focused, contextually relevant content. Jina AI made the tool accessible by

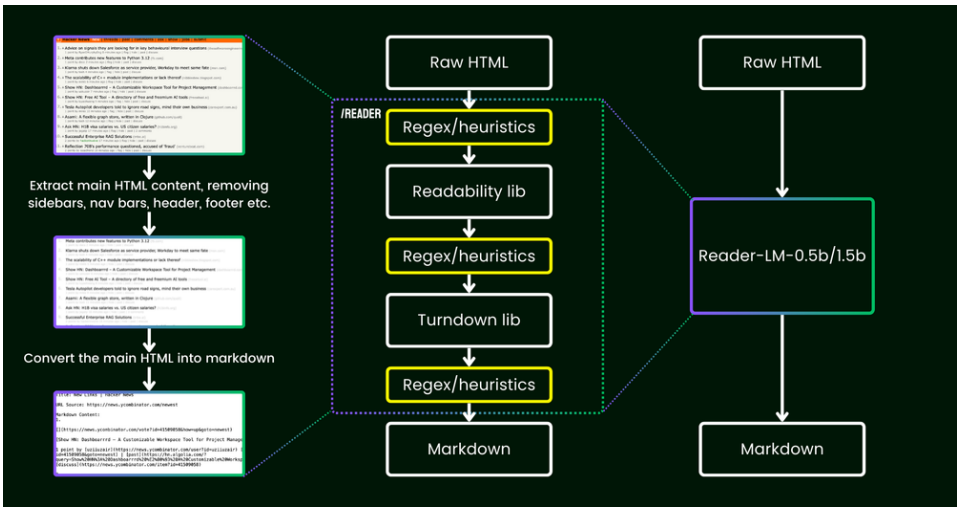

Figure 2.1: An illustration highlighting the difference between reader API and reader language model pipelines. In the reader language model, the goal is to replace the hard-coded cleaning phases found in the naive reader API by Jina, offering a more dynamic and efficient solution. Source: `www.jina.ai`

adding "r.jina.ai" before any URL, and some parameters are available through request headers. The tool first fetches the source of the webpage, then leverages Mozilla's Readability package to extract the main content and remove non-informative elements (like headers, footers and sidebars) and finally converts the cleaned HTML into markdown using regex and Turndown library.

2.1.2 Reader Language Model by Jina AI

Reader LM, developed by Jina AI, is a specialized Small Language Model (SLM) designed to process and clean raw HTML-formatted web page texts, converting them into markdown formats optimized for use with LLMs. The development of this dedicated SLM addresses limitations observed in the Reader API, particularly in cases where determining the appropriate level of detail, retaining relevant content, or accurately converting HTML to markdown posed challenges. For instance, the Reader API may inadvertently remove essential information or Turndown can struggle with content conversion.

While specific issues in HTML-to-markdown conversion can be partially

mitigated with custom regex patterns, this approach is difficult to sustain due to its high maintenance demands and limited multilingual support. In contrast, Reader LM's tailored capabilities could offer a more robust, adaptable solution, enabling better content filtering and transformation across varied languages and formats.

2.1.3 Spider by Spider.cloud

A recently introduced tool, Spider, is a web crawler developed entirely in Rust, designed to provide comprehensive webpage crawling and document cleaning services. This tool focuses on delivering a fast and efficient crawling process while producing outputs optimized for LLM compatibility, all within an integrated, all-in-one solution. Spider also incorporates small language models further to enhance its document processing capabilities' efficiency and adaptability [1].

In our project, we decided to use both Jina AI tools for two different tasks, creating a hybrid tool to crawl all the university web pages and resources.

2.2 RAG State-of-the-art

In the following sections, we will introduce some of the most widely adopted implementations of RAG systems utilized in recent research efforts.

2.2.1 Naive RAG

The Naive RAG research paradigm represents the earliest and simplest methodology developed. This approach follows a straightforward process involving indexing, retrieval, and generation, often referred to as a "Retrieve-Read" framework [11]. Naive RAG relies heavily on the user's prompt, which can be limited by its specificity or clarity, potentially impacting the quality of the generated responses.

[1] `https://spider.cloud/`

2.2.2 Advanced RAG

Advanced RAG represents an improved version of the Naive RAG system, incorporating targeted enhancements to address limitations in the naive approach, its structure is visibile in Figure 2.2. This system employs pre-retrieval and post-retrieval strategies to optimize performance and provide the LLM with the most relevant context possible. In the pre-retrieval phase, various techniques are applied to refine the user prompt and optimize the similarity search within the vector database. In the post-retrieval phase, reranking methods are used to select the most relevant results from the retrieved set [3].

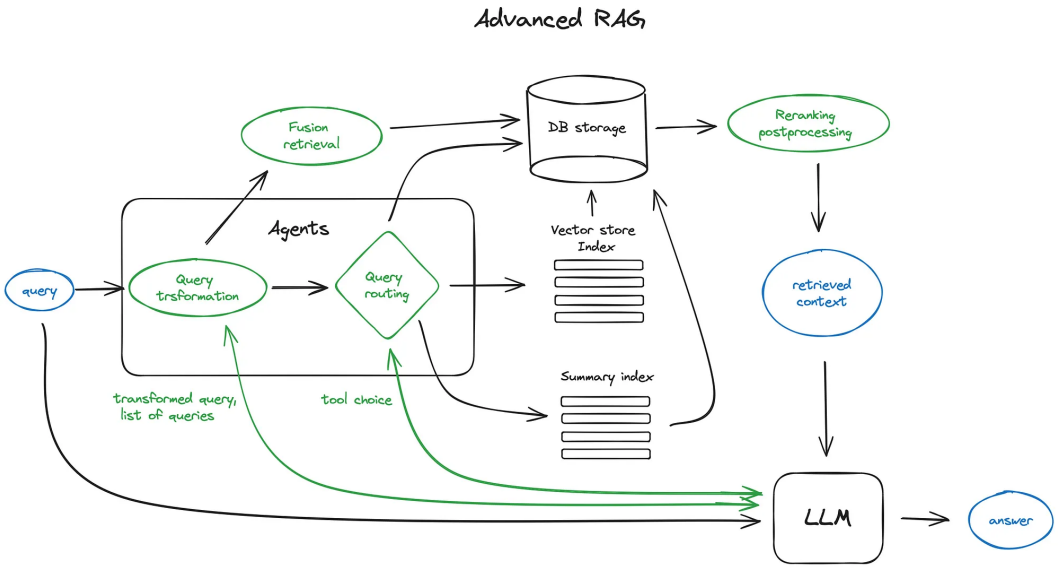

Figure 2.2: Visualization of the advanced RAG pipeline: pre-retrieval and post-retrieval strategies are implemented to maximize results obtained with the given query and vector database. Source: "Advanced RAG Techniques: an Illustrated Overview" [3].

2.2.3 Modular RAG

The Modular RAG architecture represents a flexible approach designed to enhance adaptability and versatility. This architecture consists of specialized modules that refine traditional RAG components, integrating various strategies

to optimize performance like restructured RAG modules [12] and rearranged RAG pipelines [13].

2.3 GraphRAG State-of-the-art implementations

As previously discussed in Section 1.5, although the implementations are always based on the same underlying concept, multiple implementations of the GraphRAG workflow are currently being studied and tested. In the following sections, we will introduce some of the most widely adopted implementations utilized in recent research efforts.

2.3.1 Query Generation in GraphRAG

In this implementation of GraphRAG workflow, once the graph is generated, the user query is provided to the LLM, enabling it to generate a query using Cypher Query Language (CQL)[2] to be executed on the Knowledge Graph Database. Upon execution of the query, the resulting nodes and edge triples are returned to the LLM, which subsequently utilizes this information to generate a response to the user's query. This implementation hinges on the model's capacity to interpret the graph structure and identify the most relevant nodes and edges for information retrieval [14].

2.3.2 Similarity Search Retrieval in GraphRAG

An alternative implementation performs a similarity search on the nodes within the Knowledge Graph. This approach involves generating vector indices for elements in the graph, allowing for efficient retrieval based on similarity search techniques. A critical factor to consider in this process is selecting the number of neighbouring nodes to retrieve; for each identified node, it is often beneficial to include interconnected nodes to provide comprehensive contextual information. Triplets of nodes and edges are then given to the LLM to generate the best response possible for the user query.

[2] `https://en.wikipedia.org/wiki/Cypher_(query_language)`

2.3.3 Community Clustering in GraphRAG

Despite advancements in LLMs and RAG systems, their effectiveness is frequently constrained by limited integration of entity relationships and community structures. This can diminish their ability to perform contextually enriched and accurate information retrieval, particularly for fact-checking tasks. In response to these limitations, recent studies have explored a community-oriented approach, which involves constructing a Community Knowledge Graph as the foundation of a RAG system as illustred in Figure 2.3.

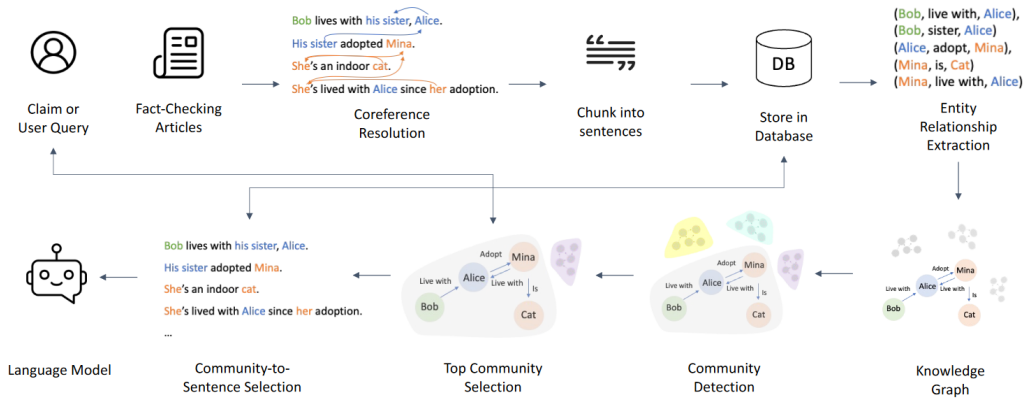

Figure 2.3: The Community Knowledge GraphRAG workflow: communities form the foundation of this technique, enhancing the previous GraphRAG implementations. Source: "CommunityKG-RAG" paper [4].

This approach involves extracting communities of entities and relationships with similar contextual attributes from source texts to populate the graph, resulting in an interconnected structure of communities composed of nodes and edges. Using relationship extraction models, entity relationships within the corpus are accurately identified and mapped, enhancing the graph's depth and cohesion. As a result, for each user query, the system can retrieve the entire relevant community, ensuring a comprehensive and contextually aligned response [4].

2.3.4 Community with Summarization in GraphRAG

This implementation evolves the previous method and focuses on leveraging community summaries to generate better results. As previously anticipated, the Language Model create a summarized contextual overview for each community. During response generation, the model formulates an answer for the original user question for each community by referencing the respective summaries. For each answer generated (one per community), a utility score is calculated on a scale from 0 to 100, reflecting the relevance and usefulness of the response in addressing the user's query. Subsequently, these responses are combined in a weighted manner, with the utility scores serving as weights, to produce a final, optimized response tailored to the user's needs [15].

Chapter 3

Method

As anticipated in previous chapters, this work utilizes Large Language Models to propose a RAG system capable of answering questions about the University of Bologna, using information available on its official website.

3.1 Dataset Creation

This first part of the project consisted of creating a complete dataset of information by crawling the University's official websites. For this work, we decided to select a subset of the website domain and gather information about three university degree programs: Bachelor's Degree in Engineering and Computer Science, Bachelor's Degree in International Development and Cooperation and Bachelor's Degree in Mathematics.

3.1.1 Crawler Pipeline

We developed a custom pipeline to crawl and process all online course resources, integrating a dedicated crawling spider with robust information-cleaning tools. For web pages' raw HTML, we implemented a specialized pipeline that uses regex-based cleaning combined with a Small Language Model to further refine and convert HTML content into Markdown, ensuring high-quality results. For PDF resources instead, we employed a tool that converts PDFs into Markdown format, effectively extracting essential information to

optimize content usability.

Domain Limitation To collect only the defined subset, we configured our focused web crawler to start from each course homepage and to limit the links it followed. Specifically, we restricted the spider to follow only URLs that began with the course-specific prefix. Links embedded within course pages that led to external resources were downloaded, but further links from those external pages were ignored. This approach ensured that all resources directly referenced on the course pages were gathered for retrieval.

3.2 Knowledge Graph

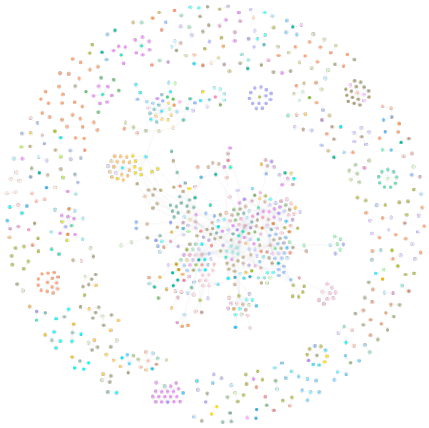

Figure 3.1: A visualization of the knowledge graph generated using our crawled courses dataset in conjunction with an open-source LLM.

Based on the promising results of GraphRAG in utilizing knowledge graphs, we decided to implement a Knowledge Graph specifically for our study case. Using an open-source LLM, we generated this graph from the website information within our retrieved dataset. The primary challenge in generating the graph was ensuring that the model consistently returned relationships in a structured format, defined by five properties: head, head_type, relation, tail, and tail_type. Various prompt versions and an output structure check were

needed to obtain an acceptable result. As shown in Figure 3.1, the LLM was unable to generate a fully connected graph, leaving numerous nodes isolated and unlinked to others. This lack of connectivity limits the graph's effectiveness for retrieving context in response to queries, reducing its overall utility.

3.3 FAQ collecting

domanda	**risposta**	**origine**
A chi è rivolta la formazione obbligatoria su sicurezza e salute?	A tutti gli studenti che, durante il percorso di studio, svolgono un tirocinio, la tesi o un'attività in un laboratorio di qualsiasi tipo.	`https://corsi.unibo.it/laurea/matematica/formazione-obbligatoria-su-sicurezza`
Tra gli esami scelti quanti si devono sostenere?	Devono essere sostenuti tutti gli esami inseriti nei 12 CFU dei corsi opzionali proposti e un numero di esami tale per arrivare a 12 CFU tra quelli scelti nell'Ateneo. Se quindi si sono scelti più di 12 CFU nella lista di corsi qualunque nell'Ateneo, non è necessario sostenerli tutti.	`https://corsi.unibo.it/laurea/matematica/faq`
Cosa sono gli OFA?	Per iscriverti al Corso è importante che tu possieda alcune conoscenze di base. La verifica delle conoscenze, le cui modalità sono dettagliate nella pagina Iscriversi al Corso, ha l'obiettivo di verificare che tu non abbia carenze significative in particolari discipline per cui è richiesta un'adeguata preparazione per affrontare con profitto il Corso. I Corsi a numero programmato prevedono la verifica dei requisiti all'interno del test di ammissione.	`https://corsi.unibo.it/laurea/IngegneriaScienzeInformatiche/come-assolvere-ofa`

Table 3.1: Frequently asked questions gathered from the official university website, with responses and source page URLs collected for evaluation and context.

To evaluate the rag system created, we gathered every frequently asked question available on the three courses' webpages with their corresponding answers. Since the questions were few and hard to identify automatically on the website pages, we decided to take this action manually. This could also be useful later on to identify questions whose answers are not available anywhere

on the website other than on the question page itself. The gathered FAQ dataset structure is show in Table 3.1.

3.4 Vector Database

In previous work, we emphasized the critical role of indexing the source dataset to achieve optimal results during the retrieval phase. We began a structured pre-processing phase after gathering all documents from the official university website. This involved removing frequently asked questions (FAQs) from the text to prevent the model from defaulting to pre-existing question-answer pairs. This step was essential for minimizing erroneous outputs from the RAG system, as it compels the model to generate responses without relying on exact matches.

Following pre-processing, we configured the vector database to populate it with document chunks, with close attention to document segmentation parameters. We employed a recursive text-splitting technique that systematically partitions the text by progressively "stronger" delimiters, beginning with paragraphs and sentences and concluding with "weaker" delimiters, such as individual words or characters. This approach helps preserve contextual coherence within each segment, which is critical for accurate retrieval and response generation.

Effective text splitting relies on carefully adjusting two main parameters: chunk size and overlap. The chunk size, which defines the maximum number of characters per segment, determines the amount of content provided to the language model. Overlap, which specifies the number of characters shared between consecutive chunks, is essential for maintaining contextual continuity. This parameter is crucial in enabling the retriever to locate the most relevant segments, ensuring the RAG system can accurately respond to user queries.

After testing, we found that a chunk size of 512 characters and a chunk overlap of 200 characters (approximately 40%) were optimal for our case. These values provided a balance between segment coherence and efficient information retrieval. Another vital aspect of effective retrieval is the addition of metadata to each chunk within the database, which enables filtering during the retrieval

phase. We included the course name as metadata for each chunk to increase retrieval specificity, facilitating targeted search and filtering.

3.5 Multiple-Choice Questions Generation

To effectively evaluate the results of GraphRAG, a new dataset comprising multiple-choice questions about the university was required. The use of multiple-choice questions, as opposed to open-ended ones, allows for precise verification of the accuracy of generated answers, enabling a quantifiable measure of correctness. Additionally, this dataset was essential for assessing the performance of direct LLM inference without the retrieval component, allowing for a comparative analysis of the proposed RAG workflow in this specific case study. To develop this dataset, unlike the approach taken for FAQs—where questions were simply collected from the website—we needed to generate questions from scratch using a language model. To carry out this task, we opted to use a model distinct from those selected for generating responses, in order to prevent potential bias. Using the same model for both question generation and answering could lead to skewed results, as the model might recognize the correct response based on its own internal knowledge and structure rather than through the retrieval phase, thus undermining the evaluation of the retrieval process.

Algorithm 1 Multiple-Choice Questions Generation.

```
Procedure GENERATE(model)
generated_questions ← []
for all doc in documents do
    for all chunk in CHUNKER(doc) do
        question_text ← model.GENERATEQUESTION(chunk)
        question_json ← JSONREPAIR(question_text)
        APPEND(generated_questions, question_json)
    end for
end for
End Procedure
```

To generate the questions, we iterated over each document in the crawled data, splitting them into chunks of a specified maximum size with an overlap. This approach ensured comprehensive question coverage across all information in the dataset, while also mitigating memory issues associated with passing lengthy resources—such as PDFs—directly to the model (Algorithm 1). By segmenting the documents in this way, we maintained focus on specific portions of the text, ensuring that the generated questions accurately reflected the full scope of the dataset. The prompt for this task was carefully designed to address multiple requirements necessary for generating valid questions. The model needed to generate questions and answers in Italian, ensure only one correct answer out of four options and produce output in a structured JSON format. Additionally, we specified that, where possible, questions and answers should incorporate statistics, dates, or numerical data. This approach aimed to create highly specific questions that would challenge the model, allowing it to generate correct responses only if the relevant context had been accurately retrieved. To ensure adherence to the desired JSON structure, we employed a library capable of converting the model's output text into a JSON object, with functionality to automatically repair any structural issues if the JSON format in the text was invalid. This approach ensured consistency and reliability in formatting, enabling seamless parsing and integration of the generated questions and answers into the dataset.

Generated Dataset Structure

Table 3.2 presents the structure of the generated multiple-choice dataset, including each question, four answer options (with one correct). We also saved a reference to the source page containing the answer for each generated question. This reference will not be provided to the model but will be retained solely for evaluation and debugging purposes.

sorgente	domanda	corretta	errata_1	errata_2	errata_3
https://corsi.unibo.it/laurea/matematica/.../bando.pdf	Qual è la data limite per l'invio dell'attestato TOLC non ricevuto automaticamente per la prima selezione?	A. 9 maggio 2024	B. 6 maggio 2024	C. 16 maggio 2024	D. 27 maggio 2024
https://corsi.unibo.it/laurea/matematica/.../bando.pdf	Per il corso di Scienze Biologiche, quale peso viene attribuito al punteggio ottenuto nella sezione di Matematica del TOLC-I?	A. 0.75	B. 0.5	C. 0.1	D. 1
https://corsi.unibo.it/laurea/matematica/.../bando.pdf/	Qual è la conseguenza penale per chi fornisce dichiarazioni false nel modulo?	A. È punito ai sensi del codice penale e delle leggi speciali in materia.	B. È soggetto a una multa di 100 euro.	C. È sospeso dall'iscrizione all'università per un anno.	D. È obbligato a ripetere l'intero corso di laurea.

Table 3.2: Dataset generated using an LLM. For each webpage crawled, multiple-choice questions were created, each with four answers, of which only the first is correct.

3.6 Answer Generation

For the answer generation task we needed to implement all the component of a RAG workflow and adapt it to the differents answers we needed. Fundamentally the only difference between open-ended and multiple-choice questions is the type of prompt given to the llm, every other element in the pipeline remain the same.

3.6.1 Models Selection

First af all we selected three differents large langue models to verify the efficiency of the system without dependency with a specific model; for each

model, we had to consider the unique structure and delimiters of its prompt template, since every model is trained using a specific one. It was crucial to choose language models capable of understanding and generating Italian, given that all documents and questions were in Italian. While the task was described in English within the prompts, we specified that responses should be generated exclusively in Italian.

3.6.2 Database Retriever

For the retrieval component, we created a database retriever responsible for selecting the most relevant documents matching the user's input query. This required using the same embedding model employed during database population to ensure consistency in document representation. Since the database is hosted locally on our server (where the RAG system also runs), we connected to it using its local network identifier.

The retriever required setting two key parameters: "fetch_k" and "k". The "fetch_k" parameter specifies the number of initial documents retrieved from the entire database based on cosine similarity. From this subset, "k" represents the final selection of documents, chosen via the Maximal Marginal Relevance (MMR) algorithm, which optimizes for both relevance and diversity to avoid excessive similarity among the selected documents. To evaluate the performance of our RAG system, we set "fetch_k" to 40 and generated answers using varying values of 'k' (4, 8, 15, 20, 25, and 30). This approach allowed us to assess the LLM's performance across both low and high document counts, providing insight into the system's effectiveness with differing levels of contextual information.

3.6.3 Generation Pipeline

To link the retrieval and generation phases, we developed a pipeline - visible in Figure 3.2 - that processes the user query as input. First, 'k' documents are retrieved based on the user query. Then, a function extracts the relevant content from these documents. The prompt is constructed by concatenating the LLM instructions, the extracted document content, and the original user

Figure 3.2: A visualization of the RAG pipeline components, where each node in the pipeline is executed for every query.

query, using appropriate delimiters to match the specific prompt structure of the chosen LLM. Once the prompt is prepared, it is passed to the LLM for response generation. Finally, the output is parsed and returned as a string.

3.6.4 Open-Ended Answers vs Multiple-Choice Answers

The pipeline described earlier is the same for both open-ended and multiple-choice questions, with the only difference being the instructions provided to the LLM. For open-ended questions, we simply specify that the model should generate an open-ended response. For multiple-choice questions, we instruct the model to respond with the letter and text of the correct option. We also include a function to shuffle the answer options for the multiple-choice scenario; this ensures that the model does not receive any implicit hints about which option is correct.

Chapter 4

Experimental Setup

4.1 Environment

Inference with LLMs is highly resource-intensive, often requiring state-of-the-art hardware. For initial tests, we utilized Google Colab's free resources (NVIDIA T4 GPU), but its memory limitations necessitated access to a more powerful setup with unrestricted time and memory. Experiments were conducted on a cluster of four GPU-equipped nodes provided by the Unibo Natural Language Processing Group, led by Prof. Gianluca Moro.

We primarily used the main server, Faretra, which is equipped with an AMD EPYC 7443 24-core processor, four NVIDIA GeForce RTX 3090 GPUs with 24GB VRAM each, and 124GB of system RAM.

To access the powerful GPUs, which are shared among university students and researchers, we utilized SLURM which is an open-source job scheduler for Linux clusters [16]. It manages and allocates computing resources, enabling efficient job submission, monitoring, and control on high-performance computing systems. For long inference runs, we used the byobu tool, which allowed us to detach from terminal sessions without interrupting the running jobs.

4.2 Virtual Environment

We use Docker and Dockerfiles to create a consistent, reproducible environment for our LLM and RAG experiments, ensure compatibility, and isolate dependencies [17]. Figure 4.1 shows the docker file used to create the generation and inference environment, which include some requirements files shown in Figures 4.2, Figure 4.3 and Figure 4.4.

```
FROM nvidia/cuda:12.4.0-runtime-ubuntu22.04
LABEL maintainer="disi-unibo-nlp"
# Zero interaction (default answers to all questions)
ENV DEBIAN_FRONTEND=noninteractive
# Set work directory
WORKDIR /
# Install general-purpose dependencies
RUN apt-get update -y && \
    apt-get install -y curl \
    git \
    bash \
    nano \
    wget \
    python3.9 \
    python3-pip && \
    apt-get autoremove -y && \
    apt-get clean -y && \
    rm -rf /var/lib/apt/lists/*
# Copy the requirements file into the container at /workspace
COPY requirements_base.txt .
COPY requirements_langchain.txt .
COPY requirements_milvus.txt .
# Install dependencies from requirements.txt
RUN pip3 install --no-cache-dir -r requirements_base.txt
RUN pip3 install --no-cache-dir -r requirements_langchain.txt
RUN pip3 install --no-cache-dir -r requirements_milvus.txt
RUN pip3 install setuptools==69.5.1
# Back to default frontend
ENV DEBIAN_FRONTEND=dialog
```

Figure 4.1: A Dockerfile used to create a stable environment for executing the RAG pipeline, from the Milvus database retrieval to the LLM generation process.

```
torch==2.1.2
datasets==2.19.1
wandb==0.17.0
tokenizers==0.15.2
tqdm==4.66.2
nltk==3.8.1
scipy==1.10.1
packaging==23.2
ninja==1.11.1.1
transformers==4.37.2
peft==0.9.0
accelerate==0.30.1
bitsandbytes==0.43.0
sentence_transformers==2.7.0
huggingface_hub==0.23.0
jinja2==3.1.0
ray==2.10.0
psutil==5.9.8
evaluate==0.4.2
rouge_score==0.1.2
nvidia-ml-py3==7.352.0
json_repair
```

Figure 4.2: requirements_base.txt text file.

```
langchain==0.1.20
langchain-community==0.0.38
langchain-core==0.1.52
langchain-text-splitters==0.0.1
langchainhub==0.1.15
langchain_experimental==0.0.58
docx2txt==0.8
pypdf==4.2.0
langchain_huggingface
sentencepiece
evaluate
rouge_score
```

Figure 4.3: requirements_langchain.txt text file.

```
milvus==2.3.5
pymilvus==2.4.1
FlagEmbedding==1.2.9
```

Figure 4.4: requirements_milvus.txt text file.

4.3 Tools utilized for website data crawling and cleaning

Our crawling phase was crucial to obtain the dataset on which the entire project based on. Our approach on implementing the pipeline was to create a custom spider utilizing scrapy open-sourced library for python which was customizable for our needs. Thanks to its customizability, we were able to create a custom process which consisted in gathering every links which had the course prefix in the url and, for every page we crawled

Grazie alla sua personalizzazione, siamo stati in grado di creare un processo ad hoc che lavora nel seguente modo: partendo dalla homepage del corso ogni link individuato viene processato in maniera ricorsiva. Per ogni link che portava al download di una risorsa PDF abbiamo utilizzato le Reader API di JINA AI per scaricarne una versione convertita in formato testuale e salvarla, chiudendo il ramo ricorsivo. Per ogni link che invece portava ad una pagina del corso, ne effettuavamo il download in formato html per poi convertirla in formato markdown utilizzando una prima fase di pulizia dell'html tramite regex, e una seconda fase di conversione utilizzando il Reader Language Model di Jina "reader-lm-1.5b" che permette un'estrazione efficace delle informazioni (Code fragments in Figure 4.5 and Figure 4.6).

```
class UniboSimpleCrawlerSpider(CrawlSpider):
    name ='unibo_simple_crawler'
    jina_prefix ="https://r.jina.ai/"
    allowed_domains =['corsi.unibo.it', 'unibo.it', "r.jina.ai"]
    start_urls =['https://corsi.unibo.it/laurea/matematica']
    rules =(
        Rule(LinkExtractor(
            allow=(r'https://corsi\.unibo\.it/laurea/matematica.*',)),
            callback='parse_course_item',
            follow=True #Every link found will be crawled and managed
        ),
    )
```

Figure 4.5: Scrapy crawler code.

```
    #Crawling of the PDF resources using Reader API
    def download_pdf_with_jina(self, response):
        full_link =self.jina_prefix +response.url
        headers ={
            "x-respond-with":"markdown"
        }
        yield scrapy.Request(full_link,
                             callback=self.save_pdf_markdown,
                             headers=headers)
...
    #Extract all pdfs links to download them separately and then save the
    #current page converting the html to markdown using Reader LM 1.5B
    def parse_course_item(self, response):
        pdf_links =response.css('a::attr(href)').re(r'.*\.pdf')
        for link in pdf_links:
            yield response.follow(link, self.download_pdf_with_jina)

        raw_html =response.body
        if isinstance(raw_html, bytes):
            raw_html =raw_html.decode('utf-8')
        cleaned_html =clean_html(raw_html, clean_svg=True,
                                               clean_base64=True)

        prompt =create_prompt(cleaned_html, self.llm.get_tokenizer())
        results =self.llm.generate(
            prompt,
            sampling_params=self.sampling_params
        )
        #Saving process
        ...
```

Figure 4.6: Custom crawling pipeline that integrates Jina API and Jina Reader LM using the Scrapy crawler.

4.4 Models

Embedding Model We decided to use the BAAI/BGE-M3 embedding model, which provides high-quality embeddings that capture deep semantic meaning,

making it ideal for tasks like search and question answering. Developed by the Beijing Academy of Artificial Intelligence, BGE-M3 is an open-source model that offers state-of-the-art performance and efficiently handles large datasets [18].

Large Language Model During our project, we need to find suitable large-language models that satisfy some requirements, such as being open-source and free to use. For the generation task we decided to use models with different characteristics such "Phi-3.5-mini-instruct" [1], "Meta-Llama-3.1-8B-Instruct" [2] and "Mistral-7B-Instruct-v0.3" [3].

- **Phi-3.5-mini-instruct**: A lightweight, instruction-tuned model from Microsoft which belongs to the Phi-3 model family [19], optimized for NLP tasks like text generation and question answering. It provides strong performance in resource-constrained environments.

- **Meta-Llama-3.1-8B-Instruct**: Meta's 8B-parameter LLaMA model fine-tuned for instruction-following tasks. It balances efficiency and performance, excelling in tasks like summarization, question answering, and text generation [20].

- **Mistral-7B-Instruct-v0.3**: A 7B-parameter model from Mistral, designed for instruction-following tasks. It delivers coherent, context-aware responses, making it ideal for conversational AI and content generation [21].

For the generation task we chose "gemma-2-9b-it" [4] which is a 9-billion-parameter model by Google, trained on web documents, code, and math. It performs well in text generation, coding, and multilingual tasks, making it efficient for smaller hardware setups [22].

[1] `https://huggingface.co/microsoft/Phi-3.5-mini-instruct`
[2] `https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct`
[3] `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3`
[4] `https://huggingface.co/google/gemma-2-9b-it`

4.5 Rag implementation

To implement the RAG pipeline, we chose the LangChain library, which is well-suited for integrating and managing workflows involving multiple LLMs and data retrieval tools from the Hugging Face ecosystem. LangChain simplifies the process of building advanced LLM applications by offering tools that manage prompt engineering and chain creation. As shown in Figure 4.7, we utilized langchain API to create text-generation and RAG pipelines.

4.6 Evaluation Metrics

To evaluate the results obtained for the open-ended questions, we use the ROUGE metric which is a common evaluation method for tasks involving text summarization and generation. ROUGE, which stands for Recall-Oriented Understudy for Gisting Evaluation, measures the quality of generated text by comparing it to reference texts. It does this by analyzing the overlap of n-grams (contiguous word sequences), word sequences, and word pairs between the generated and reference summaries. Various ROUGE implementations exist and, for our study, we selected ROUGE-1, ROUGE-2, ROUGE-L and ROUGE-Lsum.

- **ROUGE-1**: Measures unigram (single word) overlap between generated and reference texts, evaluating basic content similarity.
- **ROUGE-2**: Measures bigram (two-word sequence) overlap, capturing short phrase similarity to assess coherence.
- **ROUGE-L**: Uses the longest common subsequence (LCS) to measure the longest overlap of in-sequence words, assessing fluency and sentence structure.
- **ROUGE-Lsum**: A variant of ROUGE-L specifically tailored for summarization tasks, measuring LCS at the summary level to better capture thematic coherence.

```
#LLM text generation pipeline function
@torch.no_grad()
def get_hugging_face_model(model_id, hf_token, return_full_text =False):
    tokenizer =AutoTokenizer.from_pretrained(model_id, token =hf_token)
    model =AutoModelForCausalLM.from_pretrained(
        model_id, quantization_config=BitsAndBytesConfig(load_in_4bit=True),
        device_map="cuda", trust_remote_code =True, token =hf_token,
        torch_dtype=torch.float16)
    pipe =pipeline("text-generation", model=model, tokenizer=tokenizer,
                   max_new_tokens=1024, top_k=50, temperature=0.2,
                   return_full_text =return_full_text, do_sample =True)
    llm =HuggingFacePipeline(
        pipeline=pipe,
        pipeline_kwargs={"return_full_text":return_full_text}
    )
    return llm
...
#RAG chain creation function
def get_rag_chain(llm_model, prompt_template, retriever):
    SYSTEM_MESSAGE ="""..."""
    USER_MESSAGE ="""..."""
    ...
    prompt =PromptTemplate(
        template=final_prompt_template, input_variables=["context",
                                                "question"])
    rag_chain =(
        {
            "context": retriever | format_docs,
            "question": RunnablePassthrough()
        }
        | prompt | llm_model | StrOutputParser())
    return rag_chain
```

Figure 4.7: Langchain functions for RAG pipeline creation.

4.7 Prompts

After many iterations we crafted the best prompt for the needed task to obtain the best output from the LLMs (Table 4.2 and Table 4.1).

1 ***Question Answering Prompt***

```
### System message:
You are an AI assistant and provide answers to questions using
fact-based and statistical information when possible.
Use the following pieces of information to provide a concise answer
to the question enclosed in <question> tags.
If you don't know the answer, just say that you don't know, and don't
try to make up an answer.
For multiple-choice questions, respond strictly with one of the given
options:  A, B, C, or D, based on the given context.
IMPORTANT: Do not provide any explanations or additional text.

### User message:
Use the given context to generate an answer for the given
multiple-choice question.
IMPORTANT: Generate the answer strictly in Italian.
IMPORTANT: Use only the information you can find in the given context
to pick the correct answer.
IMPORTANT: Respond with only one of the 4 given options, including
the letter and the text of the correct response based on the text.
IMPORTANT: Do not include any explanations or additional text in your
response.

<context>
{context}
</context>

<question>
{question}
</question>
```

Table 4.1: Prompts used for the multiple-choice question-answering phase. Retrieved documents replace the {context} placeholder, and the user's question replaces the {question} placeholder.

1 ***Question Generation Prompt***

```
### System message:
You are an AI assistant specialized in generating questions and
multiple-choice answers based on provided context about university courses.
Your task is to create engaging, thought-provoking questions and a set of
answer choices that effectively test understanding of the material.
Key Requirements:
1. Generate one question and four answer choices (A, B, C, D) based on the
given context.
2. Ensure that exactly one answer choice is correct and directly supported
by the context.
3. Create three incorrect answers that are plausible but subtly misinterpret
or alter details from the context.
4. The incorrect answers should be challenging and require careful reading
of the context to distinguish from the correct answer.
5. If the context includes statistics, dates, or numerical data, incorporate
these into the question and answers, subtly modifying values for incorrect
options.
6. Phrase the question and answers to test higher-order thinking skills
rather than mere recall.
7. Write all text in Italian.
Output Format:
Return your response in a structured JSON format with the following
properties:
- question: The generated question
- correct_answer: The correct answer (labeled as A)
- incorrect_answer_1: First incorrect answer (labeled as B)
- incorrect_answer_2: Second incorrect answer (labeled as C)
- incorrect_answer_3: Third incorrect answer (labeled as D)

### User message:
Generate a question and multiple-choice answers based on the following
context about a university course.
Adhere strictly to the requirements.
<context>
{context}
</context>
Expected JSON format:
{{"question": "Your generated question about the given context",
"correct_answer": "A. The correct answer for the question, extracted from
the context",
"incorrect_answer_1": "B. Your first misleading but plausible answer here.",
"incorrect_answer_2": "C. Your second misleading but plausible answer
here.",
"incorrect_answer_3": "D. Your third misleading but plausible answer here."
}}
```

Table 4.2: Prompts used for the multiple-choice question generation phase. Each prompt included detailed instructions to guide the LLM in generating questions in the desired format. Multiple prompt versions were created as needed.

Chapter 5

Results and Discussion

5.1 Presentation of Results

In Figure 5.1 and Figure 5.2 an example of context retrieval with 'k'=4 is shown while, in Table 5.1, rouge metrics calculated between gold answer and generated answer are shown.

```
#Questions
Vengo da fuori Cesena, esistono degli alloggi o delle residenze per studenti?

#Retrieved Documents
IngegneriaScienzeInformatiche/materiale-di-orientamento-per-gli-studenti-delle-scuole-s
↪ uperiori/faq-ingegneria-e-scienze-informatiche-cesena.pdf.txt
IngegneriaScienzeInformatiche/richiesta-riconoscimento-attivita-lavorativa-o-extra-univ
↪ ersitaria-in-sostituzione-del-tirocinio/index.txt
IngegneriaScienzeInformatiche/bacheca/borse-di-studio-per-preparazione-o-approfondiment
↪ o-della-tesi-finale-all-estero/index.txt
IngegneriaScienzeInformatiche/segreterie-e-uffici/index.txt
```

Figure 5.1: Example of retrieved documents for a question.

```
#Context retrieved
diversi paradigmi dà agli studenti la capacità di adattarsi facilmente ad eventuali nuove tecnologie e/o linguaggi specifici utilizzati in ambito lavorativo e non visti durante gli studi.
# Vengo da fuori Cesena, esistono degli alloggi o
# delle residenze per studenti?
Esistono diverse possibilit à per gli studenti di trovare un alloggio a Cesena.
I primi riferimenti utili sono ER.GO, che gestisce lo studentato, al quale si accede in possesso
##########
Telefono +39 0547 338914 Orario telefonico Lunedì 11-12 Martedì 11-12 Mercoledì 11-12 Giovedì 11-12
Indirizzo Via Montalti 69 - 47521 Cesena FC Orario apertura al pubblico
Altre informazioni Prenota per email o telefono un appuntamento online o in presenza.
#### Segreteria studenti
**In cosa ti può aiutare:** Iscrizioni, cambi di corso, trasferimenti da e verso altri atenei, laurea, diploma supplement e altre procedure.
Nome referente Stefano Macrelli
##########
Borse di studio per preparazione o approfondimento della tesi finale all'estero - Ingegneria e scienze informatiche - Laurea - Cesena
##########
L'Ufficio Relazioni Internazionali collabora con l'Area Relazioni Internazionali di Ateneo nella gestione dei programmi di mobilità internazionale, fornisce informazioni sulle opportunità di studio, tirocinio e formazione all'estero per gli studenti iscritti ai corsi di studio afferenti al Campus di Cesena. L'ufficio inoltre, accoglie e supporta gli studenti stranieri in scambio, gli studenti internazionali e i docenti/personale amministrativo in visita presso la sede di Cesena.
```

Figure 5.2: Retrieved documents content that will be passed to the LLM to the augmented-generation phase.

5.1.1 Open-ended questions results

Our results for generating open-ended answers are presented in Table 5.2, which includes the average ROUGE scores calculated between each gold answer and its corresponding generated answer.

Our results for generating multiple-choice answers are presented in Table 5.3 and contains the percentages of accuracy for the chosen.

Gold Response	Response	Rouge			
		1	2	L	Lsum
Esistono diverse possibilità per gli studenti di trovare un alloggio a Cesena. I primi riferimenti utili sono ER.GO, che gestisce lo studentato, al quale si accede in possesso di requisiti di reddito e, dal secondo anno, anche di merito e Ser.In.Ar. che offre posti alloggio a condizioni agevolate. Inoltre, sono numerose le opportunità offerte dai privati, pubblicizzate dall'Informagiovani, e dalle bacheche dell'università. Puoi consultare i seguenti siti: ER.GO, Ser.In.Ar, Informagiovani di Cesena	Esistono diverse possibilità per gli studenti di trovare un alloggio a Cesena. I primi riferimenti utili sono ER.GO, che gestisce lo studentato, al quale si accede in possesso Telefono +39 0547 338914 Orario telefonico Lunedì 11-12 Martedì 11-12 Mercoledì 11-12 Giovedì 11-12 Indirizzo Via Montalti 69 - 47521 Cesena FC Orario apertura al pubblico. Altre informazioni Prenota per email o telefono un appuntamento online o in presenza.	0.41	0.38	0.40	0.40

Table 5.1: Open-ended questions results for each model across different values of the 'k' retrieval parameter. Rouge metrics are calculated between gold answers gathered from the website and generated answers.

Model	K	Rouge1	Rouge2	RougeL	RougeLsum
Phi-3.5-mini-instruct	0	0.17	0.03	0.11	0.12
	4	0.36	0.21	0.29	0.29
	8	0.36	0.22	0.29	0.29
Meta-Llama-3.1-8B-Instruct	0	0.12	0.02	0.09	0.09
	4	0.46	0.36	0.41	**0.42**
	8	**0.47**	**0.37**	**0.42**	**0.42**
Mistral-7B-Instruct-v0.3	0	0.17	0.05	0.13	0.13
	4	0.39	0.27	0.34	0.34
	8	0.45	0.34	0.4	0.4

Table 5.2: Open-ended questions results for each model across different values of the 'k' retrieval parameter. Rouge metrics are calculated between gold answers gathered from the website and generated answers.

Model	K	Accuracy (%)
Phi-3.5-mini-instruct	0	44.43
	4	63.94
	8	67.89
	15	67.79
	20	68.10
	25	69.01
	30	67.07
Meta-Llama-3.1-8B-Instruct	0	44.08
	4	69.97
	8	**71.99**
	15	71.32
	20	**71.99**
	25	69.18
	30	71.91
Mistral-7B-Instruct-v0.3	0	41.09
	4	65.08
	8	66.85
	15	65.83
	20	66.34
	25	66.92
	30	67.29

Table 5.3: Multiple-choice questions results for each model across different values of the ‘k‘ retrieval parameter. Accuracy is computed as the average value across the three-course domains.

5.2 Discussion

As visualized in Figure 5.3 and Figure 5.4, we can clearly see the effects of the retrieval phase on the correctness of the responses.

At k=0, the model generates responses based solely on its pre-trained knowledge. In this mode, the correctness tends to be lower, as the model does not have access to external, up-to-date, or specialized information that might improve its answers. Consequently, the responses may lack precision or relevance, particularly in cases where the question pertains to niche topics,

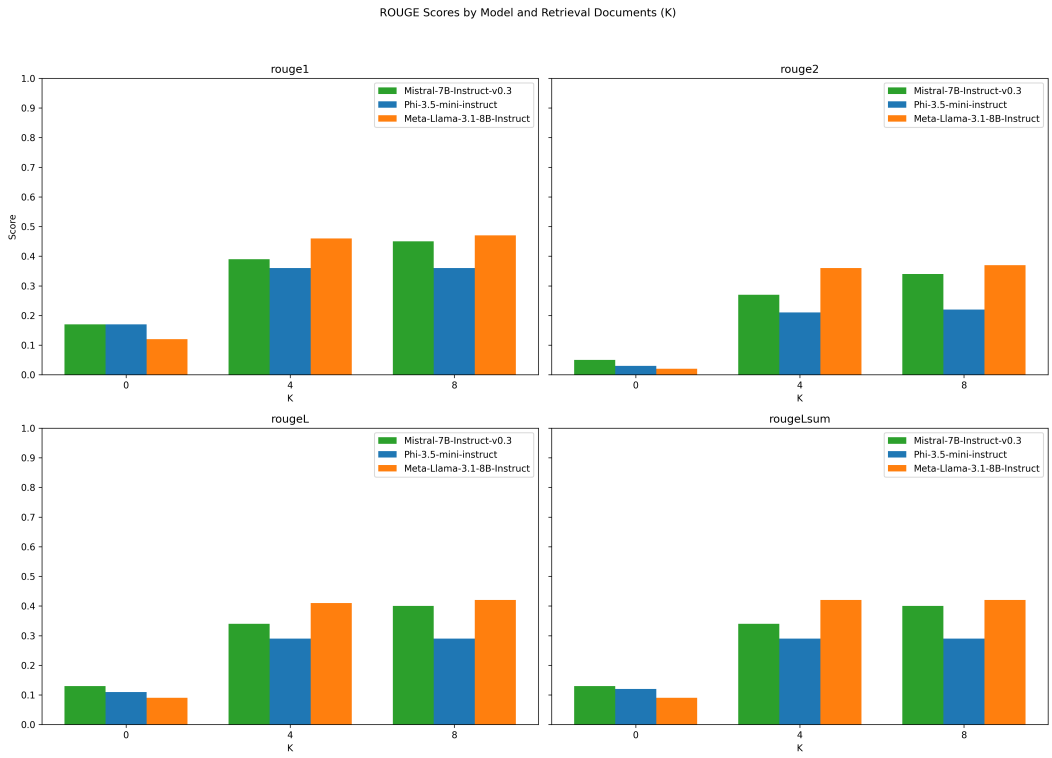

Figure 5.3: Average ROUGE accuracy across different models and varying k values.

recent developments, or highly specific information that is not contained in the model's training data.

When k increases to 4, the integration of these documents significantly improves the correctness, as the model now has access to additional, potentially up-to-date information. This retrieval process helps the model overcome gaps in its pre-trained knowledge, resulting in a noticeable improvement in the relevance and precision of the responses.

However, as k increases beyond 4—such as at k = 8, k = 15, and higher—the improvement in correctness continues, but often at a diminishing rate. For instance, while the system benefits from additional documents in the retrieval process, the increase in correctness becomes less significant as more documents are incorporated. At a certain point, around k = 20 or k = 25, the marginal benefit of adding more documents begins to decrease, and the system's perfor-

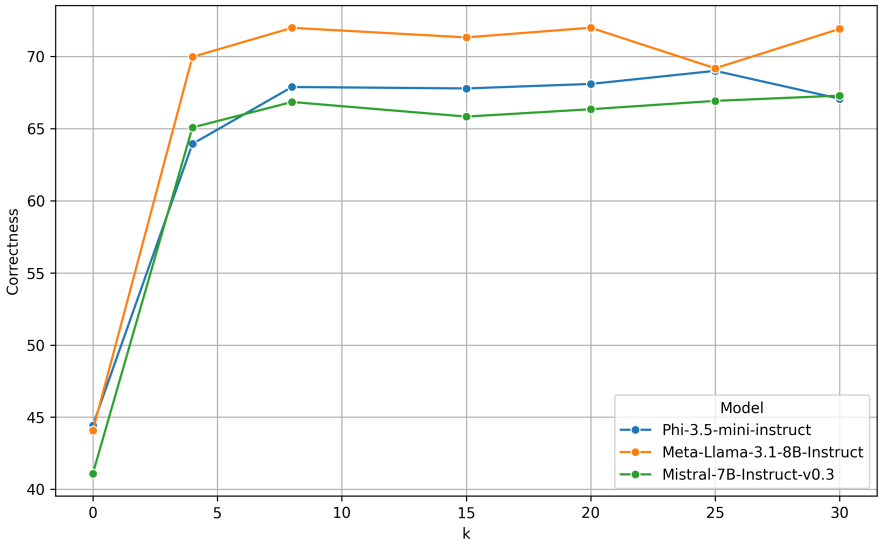

Figure 5.4: Average accuracy of models across the domains of the three courses: Engineering and Computer Science, International Development and Cooperation and Mathematics.

mance may stabilize or even slightly decline. This suggests that, after a certain threshold, retrieving more documents does not always result in a proportional improvement in the accuracy of the responses.

This plateau or slight decline in performance can be attributed to factors such as the potential redundancy of additional retrieved documents or the system's capacity to process and integrate these documents effectively. Beyond a certain point, the retrieval process may introduce more noise than useful information, affecting the model's ability to generate the most accurate response.

Conclusion and Future Work

This work underscores the effectiveness of RAG systems for question-answering tasks within a localized, highly specialized domain. We began by systematically gathering and preprocessing domain-specific information, leveraging state-of-the-art tools for website data cleaning and conversion, ensuring high-quality data for the system. This information was then embedded and stored in a vector database designed for efficient semantic retrieval.

For evaluation, we compiled a dataset of frequently asked questions from the domain, supplementing it by generating new multiple-choice questions to capture a comprehensive range of queries. To enhance precision, we used an open-source LLM to produce questions covering various details, including statistics and numerical information, creating a well-rounded benchmark dataset.

A tailored pipeline was then implemented to connect the retrieval and generation stages: user queries were matched against relevant documents in the database, which were then passed to the LLM for response generation. The pipeline supported both open-ended and multiple-choice responses, incorporating prompt customization and answer-shuffling to ensure unbiased, contextually accurate results.

In conclusion, the RAG system's performance notably exceeded that of direct LLM generation. By leveraging knowledge retrieval to ground responses, the RAG system provided contextually accurate, precise answers, highlighting its suitability for specialized domains where direct LLM responses often lack sufficient contextual grounding. This approach demonstrates RAG's potential in specialized applications, from academic knowledge bases to industry-specific information retrieval systems.

As future work, GraphRAG implementations could further enhance the system's performance by focusing on improved integration of knowledge graphs to provide deeper contextual understanding and more accurate retrieval of information. Specifically, the system could benefit from generating a more interconnected and complete knowledge graph, addressing the current limitations of isolated nodes and sparse connections. This would enhance the quality of the retrieved context for complex queries, supporting a more robust and comprehensive response generation. Another promising technology that could be used in future implementations to update the document embedding is ColPali, a new retrieval model architecture, which leverages the document understanding capabilities of recent Vision Language Models to produce high-quality contextualized embeddings solely from images of document pages [23].

Acknowledgements

The completion of this thesis represents the end of a rewarding journey enriched by the invaluable contributions of many individuals. I am grateful to all those who helped me shape this work.

I want to express my sincere gratitude to Prof. Gianluca Moro, supervisor of this thesis, who has shaped this work and provided consistent guidance and methodological and technical-scientific solutions.

I want to also thank my co-supervisors, Dott. Giacomo Frisoni and Dott. Lorenzo Molfetta, for their mentorship. The guidance and expertise of this research group were essential in navigating this emerging discipline, which was entirely new to me when I began this thesis.

Nicolas Amadori
November, 2024

Bibliography

[1] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997, 2023.

[2] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph retrieval-augmented generation: A survey. *CoRR*, abs/2408.08921, 2024.

[3] Ivan Ilin. Advanced RAG Techniques: an Illustrated Overview, 2023. URL `https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6`. Accessed: 14-11-2024.

[4] Rong-Ching Chang and Jiawei Zhang. Communitykg-rag: Leveraging community structures in knowledge graphs for advanced retrieval-augmented generation in fact-checking. *CoRR*, abs/2408.08535, 2024.

[5] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *CoRR*, abs/2402.07927, 2024.

[6] Yujuan Ding, Wenqi Fan, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on RAG meets llms: Towards retrieval-augmented large language models. *CoRR*, abs/2405.06211, 2024.

[7] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. Retrieval-

based language models and applications. In *ACL (tutorial)*, pages 41–46. Association for Computational Linguistics, 2023.

[8] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. *CoRR*, abs/2401.00368, 2024.

[9] Mohammad Abu Kausar, Vijaypal Dhaka, and Sanjeev Singh. Web crawler: A review. *International Journal of Computer Applications*, 63:31–36, 02 2013. doi: 10.5120/10440-5125.

[10] Jina AI. Reader-LM: Small Language Models for Cleaning and Converting HTML to Markdown, 2024. URL `https://jina.ai/news/reader-lm-small-language-models-for-cleaning-and-converting-html-to-markdown/`. Accessed: 14-11-2024.

[11] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models. *CoRR*, abs/2305.14283, 2023.

[12] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. *CoRR*, abs/2209.10063, 2022.

[13] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *CoRR*, abs/2305.15294, 2023.

[14] Zhuoyang Li, Liran Deng, Hui Liu, Qiaoqiao Liu, and Junzhao Du. Unioqa: A unified framework for knowledge graph question answering with large language models. *CoRR*, abs/2406.02110, 2024.

[15] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *CoRR*, abs/2404.16130, 2024.

[16] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: simple linux utility for resource management. In *JSSPP*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003.

[17] Mrs Namrata Khandhar and Mr Sagar Shah. Docker-the future of virtualization. *International Journal of Research and Analytical Reviews*, 6(2): 164–167, 2019.

[18] Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *ACL (Findings)*, pages 2318–2335. Association for Computational Linguistics, 2024.

[19] Marah I Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat S. Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Dan Iter, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xia Song, Masahiro Tanaka, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonali Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone. *CoRR*, abs/2404.14219, 2024.

[20] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[21] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023.

[22] Morgane Rivière, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen,

Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozinska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucinska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju-yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjösund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, and Lilly McNealus. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118, 2024.

[23] Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. Colpali: Efficient document retrieval with vision language models. *CoRR*, abs/2407.01449, 2024.