Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Evaluating Advanced Retrieval-Augmented Generation Techniques for Multi-hop Question Answering

A Comparative Study of Naive RAG, Recursive RAG, and Graph RAG

ZAINA RAMADAN

Evaluating Advanced Retrieval-Augmented Generation Techniques for Multi-hop Question Answering

A Comparative Study of Naive RAG, Recursive RAG, and Graph RAG

ZAINA RAMADAN

Degree Programme in Computer Science and Engineering
Date: June 4, 2025

Supervisors: Amir H. Payberah, Elsa Rimhagen
Examiner: Bo Peng
School of Electrical Engineering and Computer Science
Host company: Framna
Swedish title: Utvärdering av avancerade tekniker för hämtningsförstärkt generering vid multi-hop frågor
Swedish subtitle: En jämförande studie av Naive RAG, Recursive RAG och Graph RAG

Abstract

Retrieval-Augmented Generation (RAG) enhances large language models (LLMs) by incorporating information retrieved from external sources, such as document corpora or structured databases, to improve the accuracy and relevance of generated responses. RAG techniques range from simple, naive approaches to advanced methods using recursive retrieval or structured knowledge. While naive RAG handles basic queries well, it struggles with multi-hop QA and is prone to hallucinations. More sophisticated methods improve reasoning but typically require greater computational resources.

To assess the trade-off between accuracy and resource consumption, this thesis presents a comparative evaluation of four RAG systems: Naive RAG, Recursive RAG, KG-Graph RAG, and Community-Graph RAG. The evaluation is conducted on a multi-hop QA dataset with diverse query types, including inference, temporal, comparison, and null queries.

Each system is assessed based on generation accuracy, retrieval quality, and resource consumption. KG-Graph RAG achieves high accuracy, strong inference capabilities, and the lowest hallucination rate, while maintaining efficient token usage and moderate computational demand. Community-Graph RAG excels in temporal queries but incurs higher latency and cost. Naive RAG and Recursive RAG offer a balanced trade-off between cost and accuracy, making them suitable for general-purpose applications.

While these results offer practical guidance for selecting RAG strategies based on use-case needs, accuracy on multi-hop QA remains an area for improvement. Future work could explore hybrid systems that dynamically switch retrieval strategies based on query type to optimize both performance and efficiency.

Keywords

Retrieval-Augmented Generation (RAG), Multi-hop Question Answering (MHQA), Large Language Models (LLMs), Naive RAG, Recursive RAG, Graph RAG, Knowledge Graphs

Sammanfattning

Hämtningsförstärkt Generering (RAG) förbättrar stora språkmodeller (LLMs) genom att integrera information från externa källor, såsom dokumentsamlingar eller databaser, vilket ökar noggrannheten och relevansen i de genererade svaren. RAG-tekniker varierar från enkla, naiva metoder till avancerade tekniker som exempelvis använder sig av rekursiv hämtning. Metoder som Naive RAG hanterar enkla frågor väl, men har svårigheter med multi-hop frågor och tenderar att hallucinera i sina svar. Detta är till skillnad från mer sofistikerade metoder, som förbättrar resonemangsförmågan, men kräver mer beräkningsresurser.

I detta arbete genomför vi en jämförande studie av fyra RAG-system: Naive RAG, Recursive RAG, KG-Graph RAG och Community-Graph RAG. Utvärderingen baseras på ett multi-hop QA dataset med olika typer av frågor, inklusive inferens-, temporala-, jämförelse- och null-frågor.

Alla system utvärderas utifrån svarsnoggrannhet, kvalitet på den hämtade informationen samt resursförbrukning. KG-Graph RAG visar på hög noggrannhet, stark inferensförmåga och den lägsta graden av hallucinationer, samtidigt som det har låg tokenanvändning och måttlig beräkningsbelastning. Community-Graph RAG presterar särskilt bra vid temporala frågor, men har längre svarstid och högre token-kostnader. Naive RAG och Recursive RAG erbjuder en balanserad avvägning mellan kostnad och noggrannhet, vilket gör dem väl lämpade för allmänna användningsområden.

Sammanfattningsvis ger dessa resultat en praktisk vägledning i valet av RAG-metod utifrån användningsområde. Svarsnoggrannheten vid multi-hop QA är dock fortfarande ett område med förbättringspotential. Framtida arbete skulle kunna utforska hybrida system som dynamiskt växlar mellan hämtningstekniker beroende på frågetyp, i syfte att optimera både prestanda och effektivitet.

Nyckelord

Hämtningsförstärkt Generering (RAG), Multi-hop Frågebesvarande (MHQA), Stora Språkmodeller (LLMs), Naive RAG, Recursive RAG,

Graph RAG

Acknowledgments

I extend my sincere gratitude to my supervisor, Elsa Rimhagen at Framna, for her unwavering support, continuous interest in my work, and thoughtful input throughout the project.

I would also like to thank my KTH supervisor, Amir H. Payberah, for regularly following up on my progress and providing constructive feedback that significantly improved the quality of my work. My thanks also go to Amirhossein Layegh Kheirabadi, whose expertise helped me resolve several questions along the way.

I am grateful to many colleagues at Framna who were open to discussing my project and generously shared their insights whenever needed.

Lastly, I would like to thank my examiner, Bo Peng, for showing interest in the progress of the project and for supporting me in finalizing the work.

Stockholm, June 2025
Zaina Ramadan

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

Advanced RAG	Advanced Retrieval-Augmented Generation
AI	Artificial Intelligence
AQ	Ambiguous Questions
BERT	Bidirectional Encoder Representations from Transformers
CoRAG	Chain-of-Retrieval Augmented Generation
CoT	Chain-of-Thought
GPT	Generative Pretrained Transformer
Graph RAG	Graph Retrieval-Augmented Generation
HyDE	Hypothetical Document Embeddings
IRCoT	Interleaving Retrieval with Chain-of-Thought
LLM	Large Language Model
LoRA	Low-Rank Adaptation
LSTM	Long Short-Term Memory
MHQA	Multi-hop Question Answering
ML	Machine Learning
MMR	Maximal Marginal Relevance
Modular RAG	Modular Retrieval-Augmented Generation
MRR	Mean Reciprocal Rank
Naive RAG	Naive Retrieval-Augmented Generation
NER	Named Entity Extraction
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding

PEFT	Efficient-Fine-Tuning
QA	Question Answering
QFS	Query-Focused Summarization
RAG	Retrieval-Augmented Generation
RAPTOR	Recursive Abstractive Processing for Tree-Organized Retrieval
Recursive RAG	Recursive Retrieval-Augmented Generation
RNN	Recurrent Neural Network
ToC	Tree of Clarifications

Chapter 1

Introduction

This chapter introduces the research area of this study and the problem domain. Section 1.1 provides an overview of Retrieval-Augmented Generation (RAG), its use cases, and relevant background information necessary to understand the context of the work. Section 1.2 presents the research problem addressed in this study. Section 1.3 outlines the purpose of the work, while Section 1.4 highlights the goals to be achieved. Section 1.5 describes the research methodology adopted. Section 1.6 discusses the limitations of the study. Finally, Section 1.7 provides an outline of the structure of the thesis.

1.1 Background

Artificial Intelligence (AI) chatbots have become powerful tools, assisting millions of users every day. While general-purpose AI chatbots such as ChatGPT are readily accessible, certain use cases require systems that are tailored to specific domains. This customization is often achieved through techniques such as fine-tuning, which adapts a model to a targeted dataset. For instance, a medical or legal AI assistant is likely unable to deliver high accuracy and context awareness without fine-tuning the underlying model on domain-specific data. While fine-tuning is an effective method for tailoring chatbots to specific domains, it is often a costly and resource-intensive process, requiring large annotated datasets, significant computational power, and considerable time.

RAG is a method that has garnered significant interest from both the research community and organizations aiming to adapt AI for their business needs. RAG enhances Large Language Model (LLM) by allowing them to retrieve and incorporate additional information before generating responses, rather than relying solely on their pre-existing training data. It does so by leveraging advanced search algorithms to query external data sources such as web pages, knowledge bases, and proprietary databases. This approach is typically more cost-efficient than fine-tuning and allows for dynamic updates, ensuring the model has access to the most current and relevant information.

Several techniques have been proposed to improve the retrieval performance of RAG systems. Traditional RAG methods typically rely on similarity-based retrieval from a vector store, where the user query is matched against stored document embeddings to fetch relevant information. While effective for simple queries, Naive Retrieval-Augmented Generation (Naive RAG) often struggles with complex information needs, particularly when reasoning across multiple documents. To address these limitations, more advanced approaches have emerged. For example, some methods introduce recursive retrieval, where information is gathered iteratively from the vector store in refinement steps based on previously retrieved context. Other methods explore knowledge-graph-based retrieval, which represents relationships between entities in the text more explicitly and enables structured, multi-step reasoning. These advanced techniques aim to support more complex query handling by capturing the semantic structure of the data more effectively.

1.2 Problem

Knowledge-graph-based RAG systems (hereafter referred to as Graph Retrieval-Augmented Generation (Graph RAG)), such as Microsoft's GraphRAG [1], have demonstrated improved performance on Multi-hop Question Answering (MHQA). This is particularly of interest as MHQA tasks better reflect real-world scenarios, where user queries tend to be complex and require reasoning over multiple, distinct pieces of information. However, these improvements often come at the cost of increased resource consumption, including tokens, memory and computation time.

While the Graph RAG results are promising, current research often lacks fair and comprehensive comparisons that consider not only accuracy but also resource-related metrics such as token usage and hardware utilization. Furthermore, many studies benchmark Graph RAG methods only against Naive RAG baselines, overlooking more advanced alternatives such as Recursive Retrieval-Augmented Generation (Recursive RAG). This makes it difficult to assess the true benefits and trade-offs of using knowledge graphs. For example, one key question is whether the performance improvement offered by knowledge graph-based methods is substantial enough to justify the increased cost? Or could comparable results be achieved at a significantly lower cost, albeit with a slight drop in performance?

To ensure a comprehensive evaluation, this study considers multiple query types, including inference, temporal, comparison, and null queries (see Section 4.1). In particular, the evaluation on null queries is of interest, as it can be an indicator of each system's tendency to hallucinate information when no supporting evidence exists in the underlying data sources.

Formally the research questions are posed as follows:

1. How does the response and retrieval accuracy of Graph RAG compare to Naive RAG and Recursive RAG across different query types in multi-hop question answering tasks?

2. What are the differences in resource consumption, including response latency, token, CPU and memory usage, between Graph RAG, Recursive RAG, and Naive RAG systems?

3. What is the token cost associated with the initial construction of a knowledge graph compared to building a vector store for retrieval?

1.3 Purpose

The purpose of this thesis is to evaluate and compare the performance and resource efficiency of advanced RAG systems, with a particular focus on Graph RAG and Recursive RAG methods. By conducting a systematic comparison across the selected systems, this study aims to provide insights into the trade-offs between accuracy and resource consumption. The goal is to help guide future design decisions for organizations and researchers seeking to implement cost-effective and accurate domain-specific AI assistants.

This degree project contributes to the broader field of applied AI and Natural Language Processing (NLP) by addressing a practical gap in current research: the lack of fair, comprehensive benchmarks that consider both performance and cost. The outcomes of this study will be especially beneficial to developers, system architects, and AI researchers who must select or design retrieval systems under constrained budgets.

From an ethical standpoint, this project includes an analysis of hallucination tendencies across RAG systems, particularly in response to null queries. Reducing hallucinations in AI-generated content is critical for ensuring the trustworthiness and safety of AI applications, especially in sensitive domains like healthcare, law, and finance.

Furthermore, RAG systems inherently promote more sustainable AI practices. Unlike traditional approaches that rely on fine-tuning which often requires significant energy and computational resources, RAG systems make use of pre-trained models combined with external knowledge sources. Demonstrating the effectiveness of such systems could encourage broader adoption, thereby reducing the overall carbon footprint of AI deployment.

1.4 Goals

The primary goal of this thesis is to conduct a comprehensive and fair evaluation of several RAG methods. The aim is to shed light on the trade-offs between performance and resource usage for each system in MHQA scenarios.

The specific objectives of the project are:

- To implement or integrate systems for Naive RAG, Recursive RAG, and Graph RAG. Two systems will be evaluated for Graph RAG. The first, referred to as Community-Graph RAG, will be based on Microsoft's open-source implementation. The second, referred to as KG-Graph RAG, will be based on extracting node triplets from the constructed knowledge graph. See section 2.5.2 for more background information on the Graph RAG approaches.

- To design and execute a benchmarking framework that evaluates both retrieval and generation accuracy across multiple query types. The framework will utilize a combination of standard evaluation metrics (implemented via Python scripts) and LLM-as-a-judge techniques.
- To analyze each system's resource usage in terms of token consumption, response latency, CPU and memory usage.
- To investigate each system's tendency to hallucinate, especially in response to null queries where no relevant information exists.
- To quantify the cost of initial graph construction in terms of token usage.
- To provide actionable insights and guidelines for selecting a RAG architecture depending on accuracy requirements, data nature, and resource constraints.

1.5 Research Methodology

This thesis adopts a quantitative research methodology, focused on comparative analysis of various RAG methods. The methodology chosen is suitable for examining performance metrics, including accuracy, BERTScore Precision, BERTScore Recall, BERTScore F1, recall, latency, resource consumption, and system behavior (e.g., tendency to hallucinate). This research aligns with a positivist paradigm, which assumes that reality is objectively measurable and can be analyzed through systematic observation. The approach emphasizes empirical evidence, focusing on observable variables. By using this framework, the research can provide reliable, repeatable results that offer actionable insights.

1.6 Delimitations

This thesis project focuses exclusively on evaluating queries with single-entity or yes/no answers. It does not assess system performance for tasks requiring longer outputs, such as summarization or dialogue generation. Furthermore, while many RAG variants exist and could be of interest for evaluation, this study is limited to three approaches: Naive RAG, Recursive RAG, and Graph RAG. For the language model component, OpenAI's GPT-4o-mini [2] is the only model used. Evaluating performance using other models may yield different outcomes and is outside the scope of this project. Lastly, the evaluation is restricted to quantitative metrics. Qualitative factors such as user satisfaction or human preference judgments are not included.

1.7 Structure of the Thesis

The structure of the thesis is as follows: Chapter 2 presents relevant background information on RAG, MHQA, and the systems being compared in this study. Chapter 3 outlines the research methodology and Chapter 4 details the implementation, experimental setup and evaluation metrics. Chapter 5 presents the results of the evaluation. Chapter 6 discusses the implications of the results, compares system trade-offs, and reflects on the research questions. Chapter 7 concludes the thesis by summarizing key findings, acknowledging limitations, and proposing directions for future work.

Chapter 2

Background

The purpose of this chapter is to provide background information relevant to this thesis. Sections 2.1 and 2.2 present the concepts of NLP and LLMs. Sections 2.4 and 2.5 provide a thorough explanation of RAG and the specific techniques selected for this study. Section 2.6 explains multi-hop question answering and discusses why it poses challenges in the context of RAG. Section 2.7 describes existing evaluation approaches for RAG systems. Finally, Section 2.8 presents relevant previous work identified in the literature.

2.1 Natural Language Processing

NLP is a subfield of AI focused on enabling computers to understand, interpret and generate human language. NLP bridges the gap between human and computer communication, allowing meaningful interactions between the two. NLP is classified into two parts, Natural Language Understanding (NLU) and Natural Language Generation (NLG). NLU enables computers to comprehend and interpret human language by analyzing text and extracting concepts, entities, emotions and keywords. On the other hand, NLG focuses on producing human-like text based on structured data [3]. NLP has various applications, including Machine Translation, Text Categorization, Spam Filtering, Information Extraction, Summarization and more. Many advances have been made in the field of NLP, especially considering recent breakthroughs in Machine Learning (ML). The introduction of models like Recurrent Neural Networks (RNNs) [4] and Long

Short-Term Memory (LSTM) networks [5] allowed for better handling of sequential data, such as text, time series, financial data, speech, audio and video. More recently, models based on the Transformer architecture described in Section 2.2, such as Bidirectional Encoder Representations from Transformers (BERT) [6] and Generative Pretrained Transformer (GPT), have revolutionized NLP by leveraging attention mechanisms to capture complex dependencies in large datasets [5].

2.2 Large Language Models

LLMs are a class of deep learning models designed to process and generate text with high fluency and contextual understanding. LLMs are trained on vast amounts of text data, often sourced from publicly available corpora, allowing them to generalize across multiple domains. LLMs are built using the transformer architecture, which is a neural network architecture first proposed in 2017 by a team at Google [7]. Since their release, LLMs have demonstrated remarkable performance across various NLP tasks, ranging from text summarization and machine translation to information extraction and sentiment analysis [8]. While pre-trained LLMs perform well on open-domain tasks, they may lack the precision required for specialized applications. Fine-tuning can then be utilized to enable the model to align better with domain-specific terminology and improve factual accuracy. Fine-tuning is typically done by training an LLM on a smaller, task-specific dataset. Since models have billions of parameters (e.g., 175B in GPT-3 [9]), full fine-tuning requires extensive GPU resources and is highly inefficient and unsustainable [10]. To address this challenge, a technique, termed Efficient-Fine-Tuning (PEFT) has emerged. PEFT involves adjusting a limited number of LLM parameters while keeping the rest unchanged. Several PEFT methods exist and can be categorized into additive fine-tuning (e.g., Adapter and Soft Prompt), selective fine-tuning, reparameterized fine-tuning (e.g., Low-Rank Adaptation (LoRA)) and hybrid fine-tuning [10].

Transformers

The Transformer architecture, the main building block in modern LLMs, was introduced by Vaswani, *et al.*[7] in 2017. It rapidly became the new state-of-the-art, replacing RNNs in a wide range of NLP tasks.

RNNs are capable of leveraging sequential information by maintaining a hidden state that encodes previously seen inputs, thereby allowing the model to predict the next word in a sequence based on prior context [11]. This represented a significant improvement over traditional feedforward neural networks, which treat each input independently. However, RNNs suffer from limitations when processing longer sequences: the retained context tends to degrade as the distance between relevant tokens increases, making it difficult for the model to capture context over large spans of input data.

The Transformer architecture addresses the limitations of RNNs through the use of a self-attention mechanism, which allows the model to consider all positions in the input sequence simultaneously. Unlike RNNs, which process sequences token by token and maintain a hidden state that can decay over time, Transformers do not rely on sequential processing. Instead, they operate on the entire input sequence at once, enabling them to capture long-range dependencies more effectively. The key innovation behind Transformers is self-attention, which enables the model to dynamically assign weights to different tokens in the input based on their relevance to each other. This is achieved through Scaled Dot-Product Attention, which computes attention scores efficiently using matrix operations that scale well with larger sequences [7].

In practice, the Transformer consists of an encoder-decoder structure, where both parts are built from stacks of attention layers and feed-forward neural networks. The encoder processes the input sequence and creates a contextual representation of each token, while the decoder generates the output sequence by attending to both the encoder's outputs and previously generated tokens.

Moreover, because the architecture is fully parallelizable, it allows for significantly faster training compared to RNNs, which require sequential computation [11].

2.3 Retrieval Augmented Generation

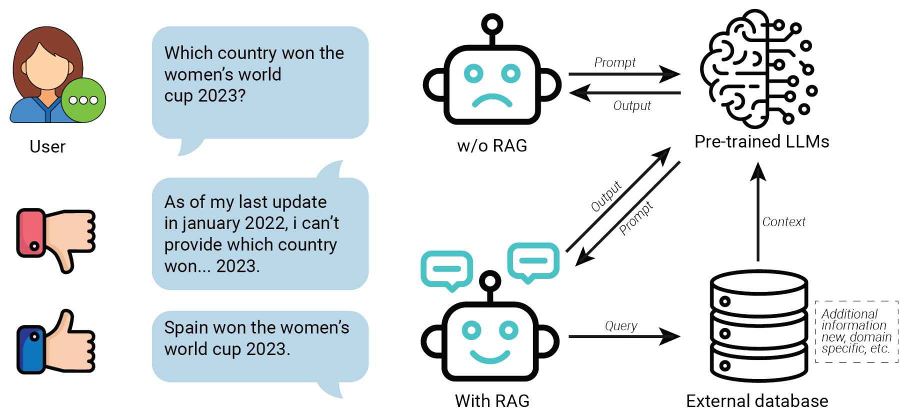

User
Which country won the women's world cup 2023?
As of my last update in january 2022, i can't provide which country won... 2023.
Spain won the women's world cup 2023.
w/o RAG
Prompt
Output
Pre-trained LLMs
Output
Prompt
Context
With RAG
Query
External database
Additional information new, domain specific, etc.

Figure 2.1: Illustration of how RAG improves the accuracy of chatbots, based on Figure 1 from the following study [12]

RAG is a technique used to enhance the capabilities of LLMs by allowing them to access and reference an external knowledge source (see Figure 2.1). Although LLMs have demonstrated strong performance across various NLP tasks, they still have certain limitations. One major issue is AI hallucinations, where models generate responses that contain incorrect or fabricated information [13][14]. Another challenge is keeping LLMs up to date with current information. The traditional approach relies on fine-tuning with new data, which is resource-intensive, making frequent updates extremely costly. RAG addresses these challenges by providing LLMs with relevant and accurate information retrieved from an external knowledge source, allowing the model to generate responses that are factually grounded and up to date [15]. RAG has been studied for domain-specific Question Answering (QA) and has demonstrated impressive results [16][17][18].

A RAG system typically consists of three main components: the retrieval component, the retrieval fusion component, and the generation component [15]. The retriever encodes inputs into embeddings and searches an indexed datastore to retrieve relevant key-value pairs. A key challenge in retrieval is balancing efficiency with quality to ensure that the most relevant information is retrieved without excessive computational overhead.

The retrieval fusion component then integrates the retrieved information using various augmentation techniques to enhance the generation process. Finally, the generator produces the final output, utilizing either standard LLMs like GPT [19] and Mistral [20] or retrieval-augmented models such as RETRO [21], which incorporate specialized modules for fusing retrieved data.

2.3.1 Embeddings

Embeddings are numerical vector representations of real-world data such as text, images, or code. These vectors map information into a multi-dimensional space, allowing ML models to process and understand complex, non-numerical data. Since ML models cannot directly interpret human language, embeddings are used to convert language into mathematical forms that preserve semantic meaning. This enables models to capture intricate relationships between different concepts.

For example, the words "smart" and "intelligent" may appear different on the surface, but their embeddings are located close to each other in the vector space since they are often used in similar contexts. This proximity allows ML models to recognize them as semantically similar, even if they are not identical. In RAG, this capability is crucial: it ensures that a user's query retrieves contextually relevant passages, even if the wording doesn't exactly match.

Transformers such as BERT [6] are commonly used to generate vector embeddings that preserve contextual meaning. These embeddings form the basis for efficient semantic search, clustering, and ranking in RAG systems.

2.3.2 Vector Store

A vector store, also known as a vector database, is a specialized data structure designed to store and retrieve high-dimensional embeddings efficiently. In traditional databases, we store and query structured data using exact matching or filtering conditions (e.g., SQL queries). However, vector databases work by storing embeddings and using similarity metrics, like Cosine similarity or Euclidean distance, to determine how close two vectors are in the embedding space.

In RAG systems, the vector store serves as the core retrieval engine. When a user submits a query, it is converted into an embedding and compared against the stored embeddings to retrieve the top-k most relevant documents, where k is the number of the retrieved documents. These documents are then passed to the language model to inform and ground the generated response.

Popular vector databases include FAISS [22], Pinecone [23] and ChromaDB [24]. ChromaDB, in particular, is an open-source, lightweight, and developer-friendly vector database often used in RAG pipelines for local development or smaller-scale applications. It integrates well with tools like LangChain [25], making it a popular choice for rapid prototyping and experimentation in LLM-based systems.

2.4 Overview of RAG Systems

RAG research can be categorized into three stages: Naive RAG, Advanced Retrieval-Augmented Generation (Advanced RAG) and Modular Retrieval-Augmented Generation (Modular RAG) [26]. Below is a brief overview of each stage, highlighting key characteristics, improvements, and challenges.

2.4.1 Naive RAG

Naive RAG is a basic form of the RAG framework that includes a process consisting of indexing, retrieval, and generation. In the indexing phase, raw data from various formats (PDF, HTML, Word, etc.) is cleaned and split into smaller chunks to accommodate the limited context window of LLMs. The chunked data is then converted into vector representations using an embedding model, and stored in a vector database. Indexing is crucial for enabling similarity searches during the retrieval phase.

In the retrieval phase, the user query is converted into a vector representation using the same embedding model as in the indexing phase. A similarity search [27] is then performed between the user query and the chunked data to retrieve the most relevant information. This retrieved information serves as additional context for the LLM during the generation step.

In the generation phase, the retrieved documents are combined with the original query to construct a prompt that guides the LLM in producing a response. A straightforward approach is to concatenate the retrieved chunks with the user query in a simple sequential format. However, more advanced prompting techniques exist, some of which have shown potential for improving system performance [28].

Naive RAG faces several challenges. The retrieval phase struggles with precision and recall, often selecting irrelevant chunks or missing crucial information. In addition, the generation process is prone to hallucinations, bias, and toxicity, leading to unreliable responses.

2.4.2 Advanced RAG

Advanced RAG enhances retrieval quality by introducing pre-retrieval and post-retrieval optimization strategies. It refines indexing through techniques like sliding windows, fine-grained segmentation, and metadata incorporation [29].

The pre-retrieval process enhances indexing and query structure through various strategies. Indexing is refined using finer data granularity, optimized index structures, metadata incorporation, and mixed retrieval. Meanwhile, query optimization aims to clarify the user's original question through techniques such as query rewriting, query transformation, and query expansion [30][31][32], along with other methods such as Hypothetical Document Embeddings (HyDE) [33].

The post-retrieval process focuses on techniques that enhance the integration of the query with the retrieved contexts. One key method is re-ranking, where a relevance score is calculated between the query and the retrieved contexts. This score is then used to reorder the contexts, prioritizing those most relevant to the query [34].

These improvements address the limitations of Naive RAG, ensuring more accurate retrieval and generation results.

2.4.3 Modular RAG

The Modular RAG system is composed of several specialized modules, each designed to handle specific tasks. While these modules operate independently, they are tightly coordinated to work together seamlessly within the overall system [35]. Six distinct modules are proposed by Gao, *et al.* [35]: Indexing, Pre-retrieval, Retrieval, Post-retrieval, Generation, and Orchestration. A key advantage of this modular approach is its flexibility where each module can be upgraded or replaced without impacting the entire system. This allows for easier customization and optimization based on the specific needs of different applications. Additionally, Modular RAG's architecture enables improved scalability, as new modules can be added or existing ones can be fine-tuned to handle larger datasets or more complex queries.

2.5 Selected Advanced Techniques

This section delves into specific advanced techniques selected and implemented in this study. First, we present relevant prior work that has investigated Recursive RAG. Then, we present two main approaches for implementing Graph RAG and discuss related work that has contributed to their development.

2.5.1 Recursive Retrieval

Recursive Retrieval is an advanced technique used in RAG pipelines, where information retrieval is iteratively refined based on the results of previous searches. Unlike traditional RAG, which retrieves a fixed set of documents based on a single query, recursive retrieval dynamically updates the query based on intermediate outputs.

One approach to implementing recursive retrieval is through Chain-of-Thought (CoT) reasoning [36], which guides the retrieval process. For instance, Interleaving Retrieval with Chain-of-Thought (IRCoT) [37] combines three components: (i) a base retriever that fetches paragraphs from a knowledge source, (ii) a language model with CoT generation, and (iii) a small set of annotated questions with reasoning steps and supporting paragraphs. Chain-of-Retrieval

Augmented Generation (CoRAG) [38] on the other hand, uses rejection sampling [39] to automatically generate the intermediate retrieval chains, eliminating the need for manual annotation.

Other approaches such as Recursive Abstractive Processing for Tree-Organized Retrieval (RAPTOR) [40] recursively embed, cluster, and summarize chunks of text to construct a tree structure with varying levels of summarization from the bottom up. Lastly, Tree of Clarifications (ToC) [41] recursively constructs a tree of disambiguations for Ambiguous Questions (AQ) through few-shot prompting and leveraging external knowledge, in an effort to generate comprehensive answers without asking the user for clarifications.

Recursive retrieval approaches for RAG have outperformed traditional retrieval methods [40] and have shown significant improvement in LLM multi-hop reasoning and QA performance for complex knowledge-intensive open-domain tasks in a few-shot setting [37].

2.5.2 Graph RAG

Graph RAG enhances the retrieval process in RAG by leveraging knowledge graphs to represent and reason over structured data. These graphs typically model entities, concepts, and documents as nodes, while relationships between them are modeled as edges. Various implementations of Graph RAG have been proposed, each leveraging the graph structure in distinct ways to improve performance and contextual understanding. In this section, we present two representative approaches: KG-Graph RAG and Community-Graph RAG.

KG-Graph RAG

KG-Graph RAG utilizes either Named Entity Extraction (NER) or more recently, an LLM with custom prompts to construct a knowledge graph from a text corpus. The process begins by chunking the text and extracting knowledge triplets from it. A triplet is a representation of a fact structured as: (entity) - [relationship] $\rightarrow$ (entity). These triplets capture the semantic relationships between entities mentioned in the corpus and form the foundation of the graph. Once extracted, the triplets are persisted in a graph database such as Neo4j [42], where each node represents an entity and each edge represents a relationship. This graph structure can then be queried to retrieve relevant information, enabling multi-hop reasoning and context-aware retrieval in downstream RAG tasks.

Several frameworks for KG-Graph RAG have been proposed, including GRAG [43], KG-RAG [44], KG^2-RAG [45], and KG-IRAG [46]. Such frameworks have shown promising results, outperforming LLM baselines, particularly in scenarios requiring detailed, multi-hop reasoning on textual graphs.

Community-Graph RAG

Community-Graph RAG constructs a knowledge graph using an LLM, and further refines it to improve retrieval effectiveness. A notable implementation of Community-Graph RAG is proposed by a team at Microsoft in the following article [1]. The high-level dataflow and the pipeline of the approach are described in detail. As a first step, the input text is split into chunks and passed into an LLM. The LLM extracts entities (e.g., person, organization, event etc.), relationships, and relevant attributes, refining the results through iterative processing to improve accuracy. The entities and their relationships are modeled as a graph and a community detection algorithm, Leiden [47] in this case, is used to partition the graph into communities. Nodes within the same community have stronger connections to one another than to the other nodes in the graph. Summaries are then generated for each detected community in the graph. The summaries are generated at either the leaf-level communities or at higher-level communities. These summaries help users understand the data's overall structure and semantics by scanning through them. Lastly, the final answer is generated through a multi-stage process leveraging hierarchical community summaries. The process follows three steps: first, community summaries are prepared by randomly shuffling and dividing them into chunks to distribute relevant information evenly. Then, intermediate answers are generated for each chunk, with the LLM assigning a helpfulness score (0-100) and filtering out low-scoring responses. The final step combines the highest-scoring intermediate answers iteratively to produce the final response. The evaluation of the implemented Community-Graph RAG framework showed substantial improvements over a Naive RAG baseline for both the comprehensiveness and diversity of answers, especially for Query-Focused Summarization (QFS) tasks.

2.6 Multi-hop Question Answering

MHQA is a task in which answering a single question requires aggregating and reasoning over multiple pieces of information, often spread across several documents [48]. The goal of MHQA is to predict the correct answer to a question that requires multiple reasoning "hops" across given contexts (text, table, knowledge graph etc).

While RAG has proven effective for single-hop question answering and open-domain QA, traditional RAG systems often fall short in multi-hop scenarios. The core limitation lies in the assumption that retrieving a handful of top-k documents is sufficient for the model to generate an accurate answer. However, in multi-hop scenarios, relevant information is often scattered across multiple sources, requiring the model to bridge facts and reason over several interconnected pieces of evidence. To address this, more advanced approaches such as Recursive RAG, which iteratively updates queries based on previously retrieved content, and Graph RAG, which builds a graph to capture relationships between entities, have emerged. These methods support more structured, stepwise reasoning and better reflect how humans navigate information to answer complex questions.

MultiHop-RAG is an MHQA dataset introduced by Tang, *et al.* [49], and is one of the first datasets designed specifically for evaluating RAG systems in the context of multi-hop queries. The motivation behind this dataset is to provide a more accurate benchmark for RAG systems in scenarios that closely resemble real-world use cases. In practice, users often submit complex or nuanced questions to RAG systems. These are questions that require synthesizing information scattered across multiple pieces of evidence. By emulating this behavior, MultiHop-RAG enables more realistic and meaningful evaluations of how well RAG pipelines can perform multi-step reasoning over retrieved content.

2.7 RAG Evaluation

The evaluation of RAG systems presents a unique challenge due to their hybrid architecture consisting of retrieval and generation components. LLMs are usually evaluated on their ability to generate accurate, fluent, and coherent text, using metrics such as BLEU, ROUGE and METEOR [50]. On the other hand, RAG evaluation must account for retrieval and generation performance both separately and jointly. To address this challenge, researchers have proposed various frameworks and metrics for RAG evaluation.

RAGAS [51] introduces a suite of metrics that enable the evaluation of RAG systems for both the retrieval and generation components, without requiring ground-truth human annotations. RAGAS evaluates RAG systems using three quality aspects: faithfulness, answer relevance and context relevance. The evaluation is fully automated by prompting an LLM and using it as a judge. For each metric, a set of instructions outlining the passing criteria is given in the form of a prompt. The LLM then assesses whether an answer or a set of retrieved documents meets the specified requirements for each metric. The results showed that the RAGAS evaluation is closely aligned with human judgments, especially for faithfulness and answer relevance. ARES [52] is another framework that uses LLM judges to evaluate RAG systems across the same three dimensions as RAGAS. ARES uses synthetic training data to fine-tune lightweight language model judges, thereby reducing reliance on manual annotations.

Other studies focus on metrics that are less frequently assessed. For instance, Zhou, *et al.* [53] propose an evaluation benchmark for RAG trustworthiness, based on six key dimensions: factuality, robustness, fairness, transparency, accountability, and privacy. Furthermore, Şakar, *et al.* [54] measure the trade-offs between vector-store based RAG accuracy, token usage, runtime, and hardware utilization.

For annotated datasets that contain a ground-truth context and a ground-truth answer, evaluation metrics such as precision, recall, F1-score, Mean Reciprocal Rank (MRR), and Hit Rate can be used to assess retrieval performance, measuring how accurately the system retrieves relevant documents [49]. Generation performance is assessed by comparing the generated response to the ground-truth answer.

2.8 Related Work

Previous studies have either proposed new techniques for improving RAG performance or presented a comparative analysis of different techniques.

Şakar, *et al.* [54] investigate the optimization of vector-store based RAG processes by evaluating various methodologies. The research involves an extensive grid-search optimization comprising 23,625 iterations, assessing multiple RAG methods (Map Re-rank, Stuff, Map Reduce, Refine, Query Step-Down, Reciprocal) across different vector stores, embedding models, and LLMs, utilizing cross-domain datasets and contextual compression filters.

Eibich, *et al.* [55] assess Advanced RAG techniques, including HyDE, LLM reranking, Maximal Marginal Relevance (MMR), Cohere rerank, Multi-query approaches, Sentence Window Retrieval, and Document Summary Index, focusing on their impacts on retrieval precision and answer similarity. The study also explores whether different combinations of these methods yield better results. HyDE and LLM re-ranking are identified as notable enhancers of retrieval precision; however, they come with increased latency and cost.

Furthermore, one study of particular relevance to this thesis project is the following [56]. The study presents a comparative analysis of Naive RAG and Graph RAG (both Community-Graph RAG and KG-Graph RAG). The study evaluates the systems using a diverse set of datasets, including a single-hop QA dataset and a MHQA dataset, with query types that overlap with those considered in this thesis. The results indicate that Naive RAG generally performs better on detailed single-hop queries, while Community-Graph RAG outperforms on multi-hop queries, particularly those involving comparison and temporal reasoning. Community-Graph RAG however showed weaker performance on null queries compared to KG-Graph RAG and Naive RAG. The study adopts an LLM-as-a-judge evaluation approach to assess the quality of generated responses.

Existing studies have repeatedly benchmarked Graph RAG against Naive RAG, potentially overlooking valuable insights that could be gained by comparing it with other advanced retrieval techniques. Moreover, studies involving Graph RAG have often neglected the critical aspect of cost and resource consumption analysis which is an important consideration when deploying real-world systems. This thesis aims to address these gaps by providing a comprehensive benchmark that evaluates both generation and retrieval accuracy. The benchmark employs metrics such as BERTScore precision, BERTScore recall, and BERTScore F1-score, in addition to incorporating an LLM-as-a-judge evaluation to offer a more nuanced and reliable assessment. It also includes metrics that are frequently omitted in prior work, such as token usage and hardware utilization.

Chapter 3

Method

The purpose of this chapter is to provide an overview of the research method used in this thesis. Section 3.1 describes the research process. Section 3.2 details the research paradigm. Section 3.3 focuses on the data collection techniques used for this research. Section 3.4 explains the techniques used to evaluate the reliability and validity of the data collected. Section 3.5 describes the experimental design. Section 3.6 describes the method used for the data analysis.

3.1 Research Process

This section outlines the key steps taken to address the problem defined in Section 1.2, from the initial stages of the project to its final implementation.

Understanding the Problem

The initial phase of this study involved an in-depth investigation into RAG to understand its existing applications and limitations. A comprehensive review of previous research was conducted to examine the frameworks, architectures, and evaluation methods commonly used in RAG-based systems. This analysis provided insights into current methodologies and performance benchmarks. Based on this understanding, a literature gap was identified, leading to the formulation of an experimental design aimed at systematically evaluating and comparing different RAG techniques.

Designing and Conducting the Experiment

To address the identified research gap, a comparative experiment was designed and conducted, focusing on three distinct RAG techniques. To establish an understanding of how a RAG system is built, a Naive RAG implementation was developed. This included deciding on and experimenting with RAG parameters and LLMs. Furthermore, a preliminary set of evaluation metrics was selected and mapped to specific RAG components to ensure a comprehensive assessment of system performance. Finally, the code for each technique was implemented, maintaining consistent parameters and environment settings across all methods to enable a controlled and rigorous comparison [57].

Results and Discussion

Once the RAG systems were implemented, each technique was executed on the dataset, and the resulting outputs were collected for analysis. Evaluation scripts were developed to assess each method systematically, using the predefined metrics. The evaluation results were then structured into tables and graphs, allowing for a clear comparison of performance across the three techniques. Finally, the findings were analyzed to draw conclusions, highlighting key insights and answering the research questions formulated in the study.

3.2 Research Paradigm

This study follows a quantitative, experimental research paradigm, as it aims to systematically evaluate and compare different RAG systems using predefined performance metrics. The research is empirical in nature, relying on controlled experiments and numerical data to draw conclusions. As such, the study aligns with the positivist paradigm, which assumes that there is a single reality that can be understood through objective measurements and systematic observations.

3.3 Data Collection

Building a RAG-based system requires access to a text corpus that can be loaded into an external knowledge source. Additionally, for evaluation purposes, a labeled dataset with ground-truth answers and retrieved evidence is needed. A suitable dataset was identified through a previous study [49] where a multi-hop dataset was produced and open-sourced on Github, along with the corresponding text corpus. This dataset was selected because it aligns with the research objectives, enabling a fair comparison of Naive RAG, Graph RAG, and Recursive RAG in handling complex, multi-step reasoning tasks.

Beyond using an existing dataset, data collection was also performed to gather results from the four RAG systems. This involved running each system on the dataset, recording the retrieved evidence, generated answers, and metric scores.

Sampling

Taking into account environmental and economic considerations, the original dataset, consisting of 619 documents and 2,556 queries, was sampled. The original dataset comprises queries from four distinct categories: temporal, comparison, inference, and null queries. To ensure a balanced representation across these categories, 50 queries of each type were randomly sampled. Furthermore, the distribution of the number of evidence required to answer queries was kept similar to the original dataset. After extracting the queries, the associated documents were also extracted using the metadata provided with each query, resulting in a final sample of 200 questions and 157 corresponding documents.

3.4 Reliability and Validity of Method and Collected Data

This section outlines the steps taken to ensure the reliability and validity of both the methods and the data used in this study. By evaluating the consistency and accuracy of the approach, the credibility of the results can be better understood and assessed.

Validity of method

Several measures were taken to ensure the validity of the methods used in this study. Firstly, a suitable dataset was selected to support fair and effective evaluation. This dataset had been used in previous research, including Microsoft's Graph RAG paper, which further supports its relevance for this project. A thorough understanding of each RAG technique was developed before any implementation began, ensuring that all components were correctly applied according to their intended design. To maintain consistency across comparisons, the same set of documents and input questions was passed to each RAG system. Additionally, the same evaluation metrics were applied to all systems to ensure that the results were directly comparable. Finally, both the system outputs and the resulting evaluation scores were manually inspected and analyzed to confirm that they aligned with expected behavior and yielded meaningful outcomes.

Reliability of method

To ensure the reliability of the methods, all experiments were conducted under consistent test settings. Each RAG system was deployed within the same controlled environment and processed the same set of input questions, eliminating variability due to differing runtime conditions. The experiments were also executed multiple times to verify that the outputs remained stable and reproducible across runs. Since outputs from language models can vary across calls, the temperature parameter was set to zero for all LLM-generated responses to ensure determinism and eliminate randomness in the generation process.

Data validity

The validity of the collected data was ensured by using standardized evaluation metrics, including accuracy, precision and recall. This captured different aspects of system performance and provided a structured and repeatable way to compare results across systems. In addition, to better account for the nature of the data and the challenges in evaluating RAG retrieval performance, an LLM-based evaluator (LLM-as-a-judge) was incorporated through the

RAGAS framework to assess outputs in a more contextually relevant manner. The results from the LLM evaluator were then compared with traditional, "static" metrics to provide a more comprehensive view of system performance and ensure the alignment between automated and human-like evaluation. By combining both traditional and LLM-based evaluation approaches, we increased the validity of the collected data and ensured that it better reflected real-world use cases.

Reliability of data

To ensure the reliability of the collected data, queries were run multiple times on each system under consistent conditions. This approach helped verify that the results were deterministic and reproducible. Any variation in the outputs was carefully noted and analyzed to ensure that the systems provided consistent responses when given the same inputs. Additionally, all systems were evaluated using the same metrics and evaluation procedures, further reinforcing the reliability of the collected data. By conducting these repeated runs and comparing the results across experiments, we were able to confirm that the data was dependable and could be trusted for further analysis.

3.5 Experimental Design

To systematically evaluate the effectiveness of different RAG techniques, ranging from naive to more advanced approaches, four RAG systems were constructed: Naive RAG, Community-Graph RAG, KG-Graph RAG and Recursive RAG. A set of carefully selected metrics was used to assess both generation and retrieval performance, as well as hardware utilization and token usage.

For generation, accuracy was used as the primary metric, as the expected answers were either yes/no or single-entity responses. Retrieval performance was evaluated using BERTScore, specifically, BERTScore precision, BERTScore recall and BERTScore F1, based on ground truth retrieved evidence. Additionally, CPU and RAM utilization were measured to assess system efficiency. Finally, input and output token counts were recorded to provide a cost analysis, and execution time per query was measured to evaluate latency.

A MHQA dataset was used to benchmark the RAG systems, ensuring identical environment settings across evaluations. After running the experiments, evaluation metrics were computed based on the model responses. Based on these results, adjustments and optimizations were applied to enhance system performance, experimenting with parameters that potentially improved answer accuracy. This ensured a fair and meaningful comparison of the three approaches.

Test Environment

This project is implemented entirely in Python, using LangChain as the development framework. For the Community-Graph RAG implementation, Nano-GraphRAG [58], a lightweight and modular framework based on Microsoft's Graph RAG implementation, was used. OpenAI's GPT-4o-mini model was employed as the generation module across all RAG systems. The constructed knowledge graph for Community-Graph RAG, along with intermediate results, was stored locally as JSON files. For KG-Graph RAG, the graph was stored in Neo4j. For the vector-store based RAG systems, ChromaDB was used as the vector store.

To assess system performance, two main scripts were developed. The first evaluated generation accuracy by cleaning up the model's output and comparing the result with the ground truth, checking for exact word matches. The second script measured retrieval accuracy using the BERTScore metrics precision, recall and F1 score. Additionally, the RAGAS evaluation framework was incorporated to better capture retrieval quality in a manner more similar to human evaluation and without requiring ground-truth labeling.

Hardware and Software Used

All RAG systems were developed using Python 3.9.6 in a virtual environment. Dependencies were managed using pip and specified in a ‘requirements.txt‘ file to ensure reproducibility. The primary libraries, tools and frameworks included LangChain, ChromaDB, and Neo4j. All LLM-related calls, including testing and evaluation using RAGAS, were made via OpenAI's API. Remaining development tasks, such as document chunking and indexing, vector store management, and non-LLM-based evaluations, were executed locally on a development machine with the following specifications:

- Device: MacBook Pro (16-inch, 2021)
- Processor: Apple M1 MAX
- Memory: 64 GB
- MacOS: Sequoia 15.3.2

3.6 Data Analysis

The generation accuracy on the QA dataset was calculated for each system to determine which RAG system had the overall best performance. This included cleaning up the generated responses and comparing them to the ground truth. The retrieved contexts were also compared against the ground-truth contexts using standard quantitative metrics implemented in Python. Additionally, the RAGAS framework was used to evaluate the retrieved contexts in a manner closer to human judgment, leveraging an LLM-evaluator. This additional evaluation approach was incorporated to guarantee a fair assessment of the retrieval performance, considering that the ground truth contexts may not always match the retrieved contexts exactly due to variability in phrasing or formulation. Furthermore, using an LLM-evaluator allows for reasoning about retrieval quality based on the user query, generated answer, and retrieved context, rather than relying solely on embedding similarity to a labeled reference.

Lastly, execution time and token usage (both input and output tokens) were measured per query for each system and recorded. This enabled us to calculate the average value per query. Additionally, CPU and RAM usage were monitored periodically (every 2 seconds) throughout the execution of all 200 queries in each system to track system performance and analyze hardware utilization.

Software Tools

The code was implemented in Visual Studio Code, utilizing a virtual environment to isolate dependencies. Git was used for version control. The following libraries were used in the experiments:

- LangChain: A framework that simplifies the development of LLM applications.
- Nano-GraphRAG: A modular and easy-to-use implementation of Microsoft's Graph RAG.
- ChromaDB: An open-source vector database optimized for storing and querying dense vectors.
- Neo4j: A graph database that supports efficient querying and visualization of graphs.
- OpenAI: Leading AI organization, offering API access to capable LLMs such as GPT 4o mini, used in this study.
- BERTScore: A semantic similarity evaluation metric for natural language generation tasks.
- RAGAS: A RAG evaluation framework that utilizes an LLM for evaluation.
- Pandas: A data manipulation and analysis library for Python.
- Matplotlib: A Python library used for visualizing data and creating a variety of graphs and plots.

Chapter 4

Implementation

This chapter details the implementation of all four RAG systems as well as the evaluation metrics. Section 4.1 presents the dataset used and the sampling strategy. Section 4.2 describes the implementation approach for each system, including code snippets and illustrations to guide the reader's understanding. Section 2.7 presents the evaluation metrics used for assessing generation and retrieval performance. Section 4.4 introduces the RAGAS framework and motivates its use. Finally, Sections 4.5 and 4.6 explain the additional metrics recorded and analyzed, namely token usage and hardware utilization.

4.1 Benchmark Dataset

This section introduces the dataset used in this study, which is specifically designed to support MHQA. It includes both the original full-scale dataset and the sampled subset used during experimentation. The dataset provides a variety of query types and evidence structures, making it well-suited for analyzing how different systems handle complex retrieval and reasoning tasks.

4.1.1 Original Dataset

The benchmark dataset used in this study is a sampled subset of the MultiHop-RAG dataset, introduced by Tang, *et al.* [49]. This dataset includes a knowledge base structured as a collection of 609 documents paired with metadata, as illustrated in Listing 4.1.

The document corpus consists of news articles published between September 2023 and December 2023, spanning a range of categories including entertainment, business, sports, health, and more. Each document contains at least 1,024 tokens in length. Additionally, the dataset includes 2,556 multi-hop queries, each paired with a ground-truth answer and corresponding supporting evidence, designed for benchmarking RAG systems. The multi-hop queries are categorized into four types: inference, comparison, temporal, and null. Each query is labeled with its corresponding type in the dataset, as shown in Listing 4.2. Below is an explanation of each query type:

- **Inference**: These queries require combining information from multiple sources in order to identify a specific entity in question, for example: Which country's prime minister visited both France and Germany in 2021?.
- **Comparison**: These queries ask for a comparative judgment between two or more entities, such as which company has more revenue.
- **Temporal**: These queries involve reasoning over time-based information, such as sequencing events and identifying durations. The answer to these queries is usually yes/no or a single temporal word such as "before" or "after".
- **Null**: These are control queries for which no correct answer can be derived from the provided corpus. They are used to evaluate the system's ability to recognize unanswerable questions and avoid hallucination.

```
{
    "title": "Fans spot Travis Kelce wearing Taylor Swift-themed
              friendship bracelet before she attended his game",
    "author": "Amber Raiken",
    "source": "The Independent - Life and Style",
    "published_at": "2023-09-26T22:05:34+00:00",
    "category": "entertainment",
    "url": "https://www.independent.co.uk/life-style/travis-kelce-
            taylor swift-friendship-bracelet-b2419065.html",
    "body": "Fans have spotted Travis Kelce wearing a friendship
             bracelet with Taylor Swift lyrics on it..."
}
```

Listing 4.1: Example of a single document from the corpus, which is stored as a JSON file. Each document contains structured metadata fields (e.g., title, author, publication date) and a body of text.

```json
{
    "query": "Between the report from 'The Independent - Life and
              Style' on September 26, 2023, regarding Taylor Swift
              and Travis Kelce, and the subsequent report from 'The
              Independent - Life and Style' on December 6, 2023,
              concerning the same individuals, was the narrative
              about their relationship consistent?",
    "answer": "Yes",
    "question_type": "temporal_query",
    "evidence_list": [
        {
            "title": "Fans spot Travis Kelce wearing Taylor Swift-
                      themed friendship bracelet before she attended
                      his game",
            "author": "Amber Raiken",
            "url": "https://www.independent.co.uk/life-style/travis
                    -kelce-taylor-swift-friendship-bracelet-
                    b2419065.html",
            "source": "The Independent - Life and Style",
            "category": "entertainment",
            "published_at": "2023-09-26T22:05:34+00:00",
            "fact": "The post came after Swift was seen
                     enthusiastically cheering him on in the box
                     seats at Arrowhead Stadium, fuelling
                     speculation that she and the athlete are
                     dating."
        }
    ]
}
```

Listing 4.2: Example of a single query from the query collection, which is stored as a JSON file. Each entry includes a query, its ground-truth answer, the query type, and the associated evidence. For readability, only one piece of evidence is shown in this example.

Sampled Dataset

Although the initial plan was to evaluate all 609 documents and 2556 queries, this proved impractical due to both time and cost constraints. Executing the full set of queries, particularly across four different RAG systems using OpenAI's GPT-4o mini, led to prolonged execution times and a high volume of API calls, which eventually triggered OpenAI's rate limiting. The most resource-intensive system was Community-Graph RAG, as it required significant computational effort for constructing the knowledge graph and executing queries on top of it. This is evident in the token-usage analysis for Community-Graph RAG presented in section 5.4.2.

For the reasons stated above, a representative sample of the original dataset was selected, resulting in an evaluation set comprising 157 documents and 200 queries. Each query type is represented by 50 queries to support category-specific analysis, particularly to assess whether any of the RAG systems are more susceptible to null query hallucinations. The distribution of the number of evidence items required to answer the queries is shown in Table 4.1, and is consistent with that of the original dataset, with most queries requiring two or three supporting documents.

Table 4.1: Distribution of the number of evidence items required to answer queries in the sampled dataset

Num. of Evidence Needed	**Count**	**Percentage**
0 (Null Query)	50	25.00%
2	74	37.00%
3	54	27.00%
4	22	11.00%
Total	200	100.00%

Sampling Strategy

To create a balanced and representative subset of the original dataset, a stratified random sampling approach was employed. The objective was to ensure equal representation of each query type while maintaining a consistent link between the sampled queries and their supporting evidence documents. The following procedure was carried out for the sampling:

1. Grouping by Query Type: The queries were grouped into the four categories (null, comparison, inference, and temporal) to

allow for random sampling from each type.

2. Uniform Sampling: 50 queries were randomly sampled from each query type group using Python's `random.sample()`.
3. Evidence Linking: Each sampled query contains one or more supporting evidence documents, identified via unique URLs. To preserve the integrity of the evaluation context, all documents referenced by the sampled queries were extracted from the original corpus.
4. Exporting the Sampled Data: The sampled queries and the associated documents were saved into separate JSON files to facilitate reproducibility and support efficient use of the dataset in evaluation tasks.

This strategy ensures that the evaluation subset maintains structural diversity, supports fair performance analysis across query types, and remains grounded in realistic multi-hop retrieval scenarios.

4.2 RAG Systems Design

This section outlines the implementation details for each of the four RAG systems. We begin by describing the parameter configurations used across the systems, followed by a deeper dive into their individual implementations, including selected code snippets to illustrate key components.

4.2.1 Selected Parameters

The choice of parameters and models in the RAG pipeline plays a critical role in the quality of the responses generated. To enable a fair and meaningful comparison between the different RAG systems, we aimed to keep core parameters, such as chunk size, embedding model, and language model, as consistent as possible across all four implementations. Below, we outline and motivate the specific parameter choices for the vector-based and graph-based RAG systems, respectively.

Vector-Based RAG Configurations

Table 4.2: Parameter configuration for vector-based RAG systems

Parameter	**Chosen Configuration**
Chunk Size	1200 characters
Chunk Overlap	100 characters
Vector Store	Chroma DB
Retrieval Method	Top-K based on Cosine similarity
Embedding Model	BGE-M3
Generation Model	GPT-4o mini

Table 4.2 presents the parameter configuration used for both the Naive RAG and Recursive RAG systems. A chunk size of 1200 tokens was selected to ensure that each chunk contained sufficient contextual information without becoming too broad. Preliminary experiments with smaller chunk sizes (e.g., 500 tokens) led to decreased accuracy, likely due to the context being too narrow. A chunk overlap of 100 tokens was used to preserve continuity across adjacent chunks, thereby reducing the risk of missing relevant information located near chunk boundaries.

Chroma DB was chosen as the vector store, enabling efficient storage of text chunks as dense embeddings and supporting similarity search between user queries and the embedded documents. For retrieval, a Top-K selection strategy based on Cosine similarity was employed, where K is a positive integer. The choice of K is discussed in more detail in Section 4.2.2 for the Naive RAG system and in Section 4.2.3 for the Recursive RAG variant.

The embedding model used was BGE-M3 [59], an open-source model accessible via the Ollama framework. For response generation, we utilized GPT-4o mini, a lightweight, cost-effective yet capable version of GPT-4 provided through the OpenAI API.

KG-Graph RAG Configurations

Table 4.3: Parameter configuration for KG-GraphRAG

Parameter	**Chosen Configuration**
Chunk Size	1200 characters
Chunk Overlap	100 characters
Storage	Neo4j (Graph Database)
Retrieval Method	Hybrid: Dense vector retrieval (Cosine similarity) + Graph retrieval (Cypher)
Embedding Model	BGE-M3
Generation Model	GPT-4o mini

Table 4.3 outlines the parameter configuration used for the KG-Graph RAG implementation. The chunk size and overlap were kept consistent with the other systems. Neo4j was chosen as the storage and query engine for the knowledge graph due to its native support for graph structures and efficient querying capabilities. To retrieve relevant context, a hybrid retrieval strategy was employed. This approach combines semantic similarity-based retrieval from a Neo4j vector index with structured graph retrieval using Cypher, Neo4j's declarative graph query language. The embedding model used for vector retrieval is BGE-M3 and the LLM, used both to build the graph and generate the final answers, is GPT-4o mini.

Community-Graph RAG Configurations

Table 4.4: Parameter configuration for GraphRAG

Parameter	**Chosen Configuration**
Chunk Size	1200 characters
Chunk Overlap	100 characters
Storage	Local File Storage (GraphML and JSON files)
Retrieval Method	Global Search / Local Search
Embedding Model	BGE-M3
Generation Model	GPT-4o mini

Table 4.4 summarizes the parameter configurations used in the Community-Graph RAG implementation. To ensure consistency

across all methods, the chunk size is set to 1200. This size strikes a balance between capturing meaningful entities and relationships for graph construction while keeping costs manageable, as building the graph involves a large number of asynchronous LLM calls. The graph is stored locally as a GraphML file, and supporting data such as community reports and intermediate results are saved as JSON files. While a database like Neo4j could be used for graph storage, local file-based storage is the default storage method and is sufficient for our use case since the system is not intended for production deployment.

To retrieve context relevant to user queries, the experiment was ran twice; once using global search and once using local search as described in Section 4.2.5. This allows for a direct comparison between the two strategies. The embedding model used is BGE-M3, consistent with the other systems. The embedding model is required for local search, where entity descriptions are embedded and used to identify suitable entry points for graph traversal (i.e., navigating through a graph structure by following edges between nodes to retrieve connected and relevant information). Finally, the LLM used for graph construction, community summarization, and final answer generation is GPT-4o mini. This model was chosen due to its strong performance and reasonable cost, both of which are critical factors given that the quality of the generated graph directly impacts the overall system performance.

4.2.2 Naive RAG Implementation

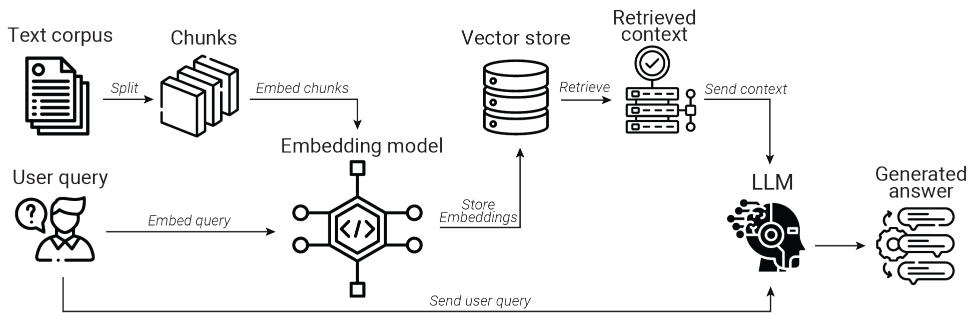


Figure 4.1: Naive RAG workflow

Figure 4.1 illustrates the Naive RAG workflow. The implementation follows this simple but effective approach to answer user queries. The process begins by chunking documents, followed by embedding and storing these chunks in a persistent vector database (see Listing 4.3).

```
documents = load_data('data/sampled_documents.json')
chunked_documents = [
    Document(page_content=chunk, metadata=document.metadata)
    for document in documents
    for chunk in text_splitter.split_text(document.page_content)
]
db = Chroma(
    collection_name="documents",
    embedding_function=HuggingFaceEmbeddings(model_name="BAAI/bge-m3
    "),
    persist_directory="chroma"
)
db.add_documents(documents=chunked_documents, ids=chunk_ids)
```

Listing 4.3: Naive RAG: Code snippet for chunking and embedding documents

When a query is received, the system performs a similarity search by comparing the embedding of the user query with the embeddings of stored document chunks in the vector database (see Listing 4.4). This process retrieves the top 10 most semantically similar text segments. We chose to retrieve the top 10 documents to strike a balance between providing sufficient contextual information and maintaining computational efficiency. Retrieving too few documents risks excluding relevant evidence, while retrieving too many can introduce noise and increase inference time without notable improvements in performance.

```
results = db.similarity_search(query, k=10)
context = "\n".join([doc.page_content for doc in results])
```

Listing 4.4: Naive RAG: Code snippet for retrieving top 10 relevant text chunks

Finally, the retrieved chunks are passed as context to the language model, which then generates the final answer (see Listing 4.5).

```
messages = [
    {"role": "system", "content": system_prompt},
    {"role": "user", "content": f"{context}\n\nQuestion:{query}\
    nAnswer:"}
]
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    temperature=0
)
answer = response.choices[0].message.content.strip()
```

Listing 4.5: Naive RAG: code snippet for answer generation

Naive RAG Prompt

```
system_prompt = """Below is a question followed by some context from
     different sources. Please answer the question based on the
    context. The answer to the question is a word or entity ONLY. If
     the provided information is insufficient to answer the question
    , respond 'Insufficient Information' only. For yes/no answers,
    answer only with  'yes' or 'no' without further explanation."""
```

Listing 4.6: Naive RAG prompt

Listing 4.6 shows the prompt used to generate responses in the Naive RAG pipeline. The prompt is carefully designed to restrict the model's output and ensure consistency in the response format. It instructs the model to answer based on the provided context, limiting the response to either a single entity or a “yes”/“no” answer, depending on the question type. Additionally, it mitigates the risk of hallucination by directing the model to respond with "Insufficient Information" if the context is inadequate to answer the query. By limiting the format of the generated answer, we simplify the evaluation process, as all outputs follow a consistent structure that is easy to parse and extract.

4.2.3 Recursive Retrieval RAG Implementation

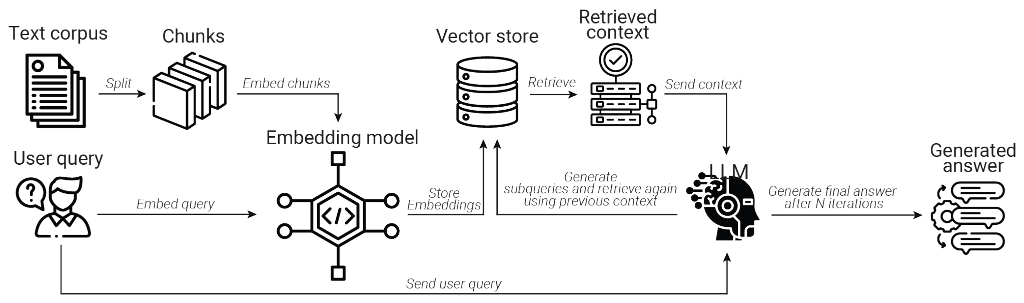


Figure 4.2: Recursive Retrieval RAG workflow

Figure 4.2 illustrates the workflow of a RAG system that incorporates recursive retrieval. The initial steps including chunking, embedding, and storing documents in the vector database are identical to the Naive RAG pipeline and reuse the same underlying code. The key difference lies in the retrieval strategy. While the naive approach performs a single retrieval step before passing the context to the language model for generation, the recursive variant executes N iterations, where N is a positive integer, retrieving up to K document chunks in each round. In every iteration, the language model receives the original query along with the previously retrieved documents and generates sub-queries intended to guide subsequent retrievals more effectively. In this implementation, we perform five retrieval iterations (N = 5), where in each iteration, up to seven chunks (K = 7) can be fetched. The retrieval loop is terminated early if no new documents are found in an iteration. To compute the average number of retrieved documents per query, we ran the retrieval process for all queries in the dataset. For each query, we counted the number of unique documents retrieved across all iterations. We then summed these counts and divided the total by the number of queries. Using this method, we found that the average number of retrieved documents per query was 10.8. Listing 4.7 presents the code used for performing recursive retrieval. Listing 4.8 illustrates the generation of sub-queries, which are formulated based on the initial query and the context accumulated during previous retrieval iterations.

```python
def recursive_retrieval(query, steps=5):
    for _ in range(steps):
        docs = db.similarity_search(query, k=7)
        if not docs:
            break
        context = update_context(docs)
        if not context:
            break
        sub_questions = generate_sub_questions(context, query)
        if not sub_questions:
            break
        query += "\n" + "\n".join(sub_questions)
    return final_documents
```

Listing 4.7: Recursive Retrieval: Code snippet for performing five iterations of retrieval

```python
def generate_sub_questions(context, initial_query):
    prompt = f""" Based on the following retrieved documents,
    context, and initial query, generate 2-3 specific sub-questions
    that:
    1. Focus on aspects not covered in the current context
    2. Help answer the initial query more comprehensively
    3. Are specific and targeted"""
    messages = [
        {"role": "system", "content": prompt},
        {"role": "user", "content": f"""Initial Query:{initial_query
    }\n\nCurrent Context: {context}\n\nSub-questions:"""}
    ]
    response = client.chat.completions.create(model="gpt-4o-mini",
    messages=messages,
    temperature=0)
    questions = response.choices[0].message.content.strip().split("\
    n")
    return questions
```

Listing 4.8: Recursive Retrieval: Code snippet for generating sub-queries based on previous context

Once the retrieval process is complete, the accumulated context is provided to the language model alongside the original query to generate a final response. This is done using the same prompt described in Listing 4.6.

4.2.4 KG-Graph RAG Implementation

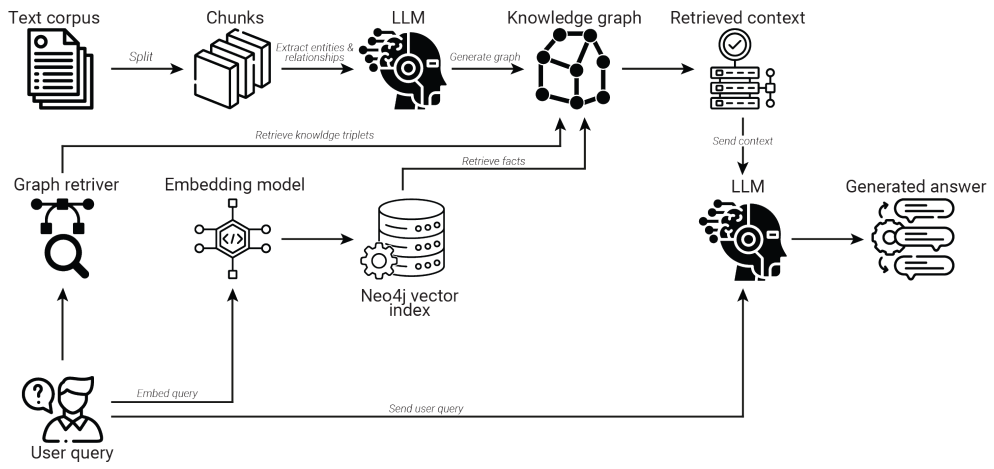


Figure 4.3: KG-Graph RAG workflow

Figure 4.3 illustrates the overall workflow of the KG-Graph RAG implementation. The process begins by splitting the document corpus into smaller text chunks. These chunks are then passed to an LLM, which extracts entities and relationships to form a structured representation of the content. This structured output, referred to as graph documents, encapsulates the semantic relationships between entities and serves as the basis for constructing the knowledge graph. The graph documents are finally ingested into a Neo4j database, enabling efficient storage and retrieval. Listing 4.9 shows the code used to build the knowledge graph using LangChain's convert_to_graph_documents function.

```
def build_graph():
    docs = chunk_documents()
    llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
    llm_transformer = LLMGraphTransformer(llm=llm)
    graph_docs = llm_transformer.convert_to_graph_documents(docs)
    graph.add_graph_documents(graph_docs, baseEntityLabel=True,
   include_source=True)
```

Listing 4.9: KG-Graph RAG: Code snippet for knowledge graph construction.

The resulting knowledge graph comprises 10,793 nodes and 28,346 relationships. An excerpt of this graph structure is visualized in Figure 4.4, illustrating the types of nodes and relationships identified during the graph construction process. As shown in the figure, nodes are color-coded according to their type. Purple nodes represent document chunks, which include associated metadata such as publication date, source, title, and URL, as outlined in Table 4.5. The remaining nodes correspond to extracted entities, each containing an identifier and a value (e.g., an entity name).

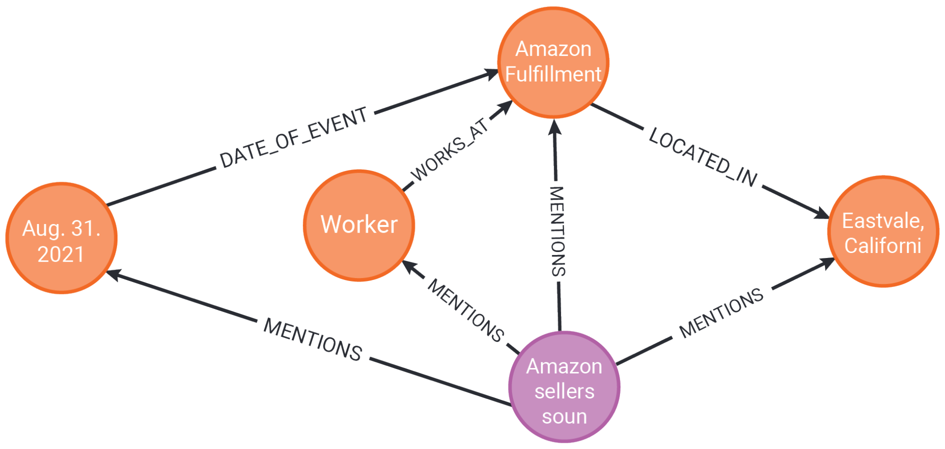


Figure 4.4: KG-Graph RAG knowledge graph excerpt

Table 4.5: Metadata of a sample document node in the knowledge graph

elementId	4:806af3a9-2661-436a-bde0-3d2b6be0f75d:89
id	89
embedding	`[0.0158596523, -0.038642671, 0.01054068, ...]`
document_id	`d23d0211071a902354f77a69ec7d0b2f`
published_at	`2023-10-06T21:31:00+00:00`
source	CNBC | World Business News Leader
text	A worker sorts out parcels in the outbound dock at the Amazon fulfillment center in Eastvale, California, on Aug. 31, 2021.
title	Amazon sellers sound off on the FTC’s ’long-overdue’ antitrust case
url	`https://www.cnbc.com/2023/10/06/amazon-sellers-sound-off-on-the-ftcs-long-overdue-antitrust-case.html`

When a query is submitted, hybrid retrieval is used to fetch relevant context. Firstly, the graph is queried to retrieve nodes whose entity labels match the entities in the user question. The nodes and their edges are returned in the form of triplets (source, relation, target). Furthermore, a Neo4j vector index is initialized using the embeddings associated with the document nodes in the knowledge graph. To retrieve additional relevant context, the user query is embedded and compared against this index using similarity search. This process returns the most semantically similar document chunks, thereby complementing the graph-based retrieval with dense vector retrieval (see Listing 4.10).

```
def full_retriever(question):
    graph_data = graph_retriever(question)
    vector_data = [
        {
            "source": "Neo4j␣Vector",
            "fact": el.page_content
        }
        for el in vector_retriever.invoke(question)
    ]
    return graph_data, vector_data
```

Listing 4.10: KG-Graph RAG: Code snippet for hybrid retrieval

In the last step, the vector and graph data are passed along with the user query to the language model, in order to generate the final answer, using the same prompt described in Listing 4.6.

4.2.5 Community-Graph RAG Implementation

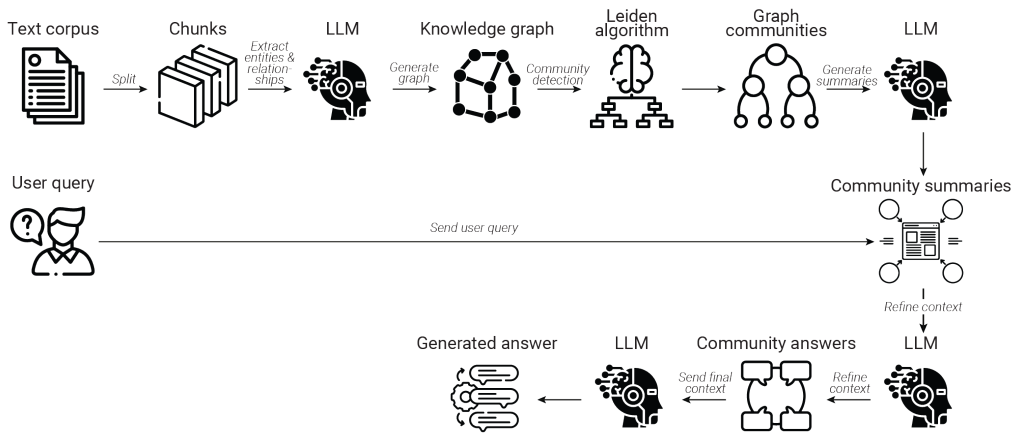


Figure 4.5: GraphRAG workflow

Figure 4.5 illustrates the workflow of the Community-Graph RAG implementation, which uses a graph-based retrieval strategy to capture and exploit relationships between entities and concepts across documents. We use a modified version of the Nano-GraphRAG implementation [58] to build and query the graph. This library is based on Microsoft's Graph RAG implementation. Nano-GraphRAG enables lightweight yet effective graph construction and supports retrieval methods that traverse the graph to find semantically and structurally relevant contexts.

During the indexing phase we construct a knowledge graph by utilizing an LLM. The LLM is prompted to extract semantic entities, such as persons, organizations, events, and other relevant types, from pre-processed and chunked documents. These entities, along with the relationships inferred between them, are used to build a structured knowledge graph. To identify meaningful clusters within the graph, the Leiden algorithm is employed to detect communities by grouping strongly connected nodes. Then the graph communities are used to generate community summaries. The resulting knowledge graph, constructed from our sampled dataset, comprises 29,632 nodes and 19,180 relationships. An excerpt of the resulting graph structure, showcasing a subset of extracted nodes and relationships, is provided in Figure 4.6.

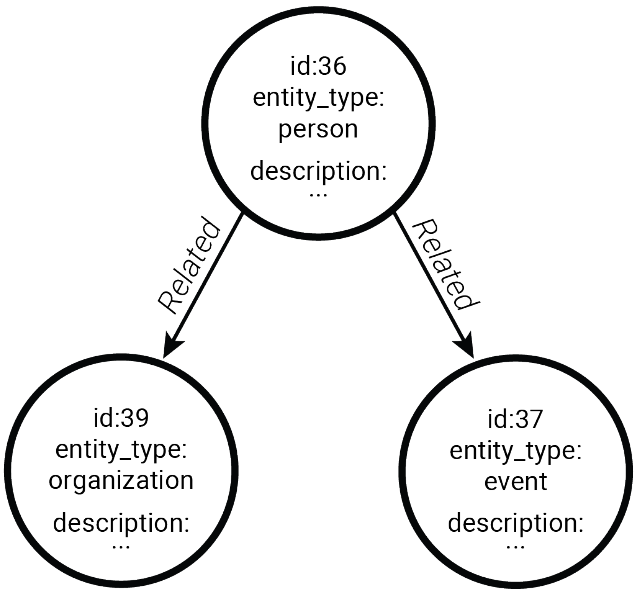


Figure 4.6: An excerpt of the constructed knowledge graph

Furthermore, Table 4.6 details an example of the metadata stored at each node. `elementId` and `id` are used to uniquely identify the node. The `clusters` field lists the communities or groups that the node belongs to across different clustering levels. The `description` field offers a textual summary of the entity, derived directly from the source documents. The `entity_type` indicates the type of entity represented by the node. Lastly, the `source_id` points to the original chunk or section of the document from which the entity and its associated metadata were extracted.

Table 4.6: Community-Graph RAG node metadata

elementId	4:8fd013fd-1ef5-48b6-bcb1-94c7d0392319:36
id	36
clusters	["level": 0, "cluster": 3, "level": 1, "cluster": 81]
description	"Fred Ruckel is a seller on Amazon, known for his product ’Ripple Rug’..."
entity_type	PERSON
source_id	chunk 3552f1f05c8dca81ade9afcf7cf6b0cd

During the retrieval and generation phase, the community summaries serve as a basis for retrieving relevant contextual information. This context is then refined through a multi-stage process involving an LLM. The refinement occurs by first generating intermediate responses at the community level, which are subsequently aggregated and polished to form a final global output. Listing 4.11 demonstrates the code used to generate answers. To retrieve context, we experimented with both global and local search to evaluate the strengths and limitations of each approach. Global search is designed for queries that require reasoning across the entire dataset, leveraging community summaries extracted from the knowledge graph. In contrast, local search is tailored for entity-centric questions, providing detailed information about specific people, organizations, or events. This is achieved by identifying relevant entities and expanding outward to include their neighboring nodes and associated concepts. The desired search mode can be specified as a parameter to `query()`.

```
def query(query):
    rag = GraphRAG(
        working_dir=WORKING_DIR,
        best_model_func=llm_model_if_cache,
        cheap_model_func=llm_model_if_cache,
        embedding_func=ollama_embedding,
    )
    answer, context = rag.query(
        query, param=QueryParam(mode="global")
    )
    return answer, context
```

Listing 4.11: Community-Graph RAG: Code snippet for querying and answer generation

Community-Graph RAG Prompt

```
system_prompt = """You are a helpful assistant responding to
    questions about a dataset by synthesizing perspectives from
    multiple analysts.
---Goal---
Generate a **single word or entity** that answers the user's
    question based on the context provided in the reports.

For yes/no questions, answer with **yes** or **no** only, without
    further explanation.

The analysts' reports are ranked in the **descending order of
    importance**, and you should focus on the most relevant and
    informative data provided.

Remove any irrelevant information from the analysts' reports before
    deriving the final answer. Ensure the answer is concise and
    directly related to the question.

Do not include information where the supporting evidence for it is
    not provided, answer instead with **Insufficient information**.
    """
```

Listing 4.12: Community-Graph RAG prompt

Listing 4.12 details the prompt used to generate the final global answer. Similarly to the Naive RAG prompt, this prompt was modified to restrict the model output to single entity answers, yes/no answers or "Insufficient information". In addition, there is a specific prompt (see Listing 4.13) that is designed to respond with "Insufficient information" in the case that no relevant context or graph relationships are found.

```
PROMPTS["fail_response"]="Insufficient␣information"
```

Listing 4.13: Community-Graph RAG: No relevant context found prompt

Since the Community-Graph RAG implementation relies on an LLM for multiple tasks, including entity extraction, context retrieval, and context refinement, various specialized prompts are employed at different stages of the pipeline. This design introduces flexibility and opens up opportunities for prompt experimentation and fine-tuning. Prompts can be adapted or optimized to better suit a particular dataset, domain-specific use case, or performance requirement, making Community-Graph RAG a highly customizable framework.

4.3 Evaluation Metrics

In this section, we describe the metrics used to evaluate both the generation and retrieval components of our systems. Generation metrics evaluate the accuracy of the model's output based on the retrieved context, while retrieval metrics assess how well the retrieved context matches the ground-truth. During context retrieval and response generation, outputs (e.g., answer and retrieved context) were saved in the same format as the ground-truth dataset (see Listing 4.2). Maintaining a consistent format simplifies the evaluation process and enables direct comparison between generated answers and reference labels.

4.3.1 Generation Metrics

To assess generation quality, we use the accuracy metric, which compares the generated answer to the ground-truth label.

Accuracy

To evaluate each system's ability to produce correct answers, we measure the accuracy of its responses. Since the model is constrained to generate yes/no answers, single entities, or the string "Insufficient Information", the output space is well-defined and discrete. This makes it straightforward to extract the predicted answer and compare it against the ground truth. For each evaluated example, a correct prediction is recorded as a 1, and an incorrect one as a 0. The overall accuracy is then computed using the following formula:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N}[\hat{y}_i = y_i] \tag{4.1}$$

where:

- N denotes the total number of evaluated examples.
- $\hat{y}_i$ is the predicted answer for the i-th example.

- y_i is the corresponding ground-truth answer for the i-th example.
- $[\cdot]$ is the Iverson bracket [60] returning 1 if the predicted answer matches the ground truth, and 0 otherwise.

4.3.2 Retrieval Metrics

Evaluating RAG retrieval performance is a challenging task that requires careful consideration of several factors. Upon examining the retrieved contexts from each system and comparing them to the ground-truth contexts, it became clear that exact word matching or fuzzy string matching would not yield satisfactory results. This is largely due to the nature of the retrieved content, which can differ significantly from the ground truth depending on factors such as chunk size, the embedding model used, and other pre-processing steps. In particular, the context produced by Community-Graph RAG has undergone multiple rounds of LLM-based summarization, making it even less likely to match the ground truth verbatim. Therefore, a metric capable of evaluating semantic similarity rather than surface-level overlap was required. BERTScore [61] was selected as a suitable evaluation metric, as it compares the contextual meaning of the retrieved content to that of the ground truth using embeddings from pre-trained language models.

BERTScore Precision

BERTScore Precision measures how well the retrieved context $\hat{x}$ semantically aligns with the ground truth context x. For each token in $\hat{x}$, the maximum Cosine similarity with any token in x is computed. Precision is then defined as the average of these maximum similarities. Formally, BERTScore Precision, or P_{BERT} is computed using the following formula:

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^{\top} \hat{x}_j \tag{4.2}$$

where:

- $\hat{x}$ is the set of tokens in the retrieved context.

- x is the set of tokens in the ground-truth context.
- $x_i^\top \hat{x}_j$ denotes the Cosine similarity between the contextual embeddings of the i-th token in x and j-th token in $\hat{x}$.

A higher BERTScore Precision indicates that the retrieved content contains tokens that closely match the reference in meaning.

BERTScore Recall

BERTScore Recall measures how well the ground truth context x is covered by the retrieved context $\hat{x}$. For each token in x, the maximum Cosine similarity with any token in $\hat{x}$ is computed. Recall is then defined as the average of these maximum similarities. Formally, BERTScore Recall or R_{BERT} is defined as:

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^\top \hat{x}_j \tag{4.3}$$

where:

- x is the set of tokens in the ground-truth context.
- $\hat{x}$ is the set of tokens in the retrieved context.
- $x_i^\top \hat{x}_j$ denotes the Cosine similarity between the contextual embeddings of the i-th token in x and the j-th token in $\hat{x}$ respectively.

A higher BERTScore Recall indicates that the retrieved content successfully covers the meaning expressed in the ground truth.

BertScore F1

BERTScore F1 combines Precision and Recall to provide a balanced measure of how well the retrieved context $\hat{x}$ aligns with the ground truth context x. It is computed as the harmonic mean of P_{BERT} and R_{BERT}. Formally, the BERTScore F1 or F_{BERT} is defined as:

$$F_{\text{BERT}} = 2 \times \frac{P_{\text{BERT}} \times R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{4.4}$$

where:

- P_{BERT} is the BERTScore Precision.
- R_{BERT} is the BERTScore Recall.

4.4 RAGAS - LLM as a Judge Evaluation

Using an LLM to evaluate how well the retrieved context in a RAG system supports answering a user query has become a popular approach, with several RAG studies adopting it. This method enables flexible, human-like evaluation of system performance. Perhaps more importantly, it allows for evaluation of both the retrieval and generation steps without requiring access to an annotated dataset. Instead, the LLM uses only the query, the generated answer, and the retrieved contexts as input. Well-crafted prompts are then employed to guide the LLM in its evaluation.

Whilst BERTScore can measure retrieval performance by assessing semantic similarity against ground-truth labels, it lacks awareness of the specific context and intent of the user query. Moreover, it requires access to annotated datasets, which limits its applicability in scenarios where labeled data is unavailable or costly to produce.

To address this limitation and provide another evaluation dimension, we employed the RAGAS framework [51], which leverages LLMs to assess the quality of RAG pipelines across multiple dimensions. RAGAS adopts an LLM-as-a-judge paradigm to evaluate whether the retrieved context adequately supports the generated answer, even when surface-level string similarity is low. The metrics used from RAGAS were: context precision without ground-truth answer reference, context precision with ground-truth answer reference, and context recall, as seen in listing 4.14. We compute retrieval precision both with and without access to the ground-truth answer using the RAGAS framework. This allows for an assessment of whether RAGAS is able to accurately evaluate the retrieval performance using only the query, answer and retrieved context, i.e., without explicit access to ground-truth labels.

```
result = evaluate(
    dataset = dataset,
    metrics=[
        context_precision_without_ref,
        context_precision_with_ref,
        context_recall
    ]
)
```

Listing 4.14: RAGAS metrics

4.5 Token Usage

Requests sent to LLMs are typically broken down into tokens, which represent units of text such as word fragments or characters. On average, one token corresponds to approximately four characters in English. Token usage is a critical factor in cost estimation, as providers such as OpenAI charge based on the number of tokens processed, both input tokens (e.g., prompt and retrieved context) and output tokens (e.g., the generated response).

To evaluate the efficiency and economic feasibility of the different RAG systems, token usage was recorded for each query processed by each system. The OpenAI API provides detailed token usage statistics, including the number of input, output, and total tokens, as illustrated in Listing 4.15.

For Community-Graph RAG, token counts were accumulated across all API calls made during the execution of a single query to account for its multi-step retrieval and generation process. For Recursive RAG, token usage related to the generation of sub-queries was also included to ensure an accurate and comprehensive measurement.

In the cost analysis (see Section 5.6), the recorded tokens per query were used to estimate the cost of using each system in a realistic deployment scenario. This analysis enables a comparative assessment of the trade-off between system performance and cost-efficiency.

```
input_tokens = response.usage.prompt_tokens
output_tokens = response.usage.completion_tokens
total_tokens = response.usage.total_tokens
```

Listing 4.15: Recording token usage

4.6 Hardware Utilization

To complement the cost and performance analysis, system-level resource usage was also monitored. Specifically, CPU and RAM utilization were recorded at two-second intervals throughout the execution of all 200 queries for each system. The Python library, `psutil`, was used to measure CPU and memory usage. This monitoring provides insight into the computational overhead and memory demands associated with each RAG approach. Furthermore, tracking hardware utilization helps identify potential bottlenecks, assess scalability, and evaluate the suitability of each system for deployment in environments with limited computational resources.

```
cpu_usage = psutil.cpu_percent(interval=2)
ram_usage = psutil.virtual_memory().percent
```

Listing 4.16: Recording Hardware Utilization using psutil in Python

Chapter 5

Results and Analysis

This chapter presents the results of the evaluation on the multi-hop dataset for all four RAG systems. Section 5.1 reports the generation accuracy of each system, including a detailed breakdown by query type. Section 5.2 presents the BERTScores for the retrieval step, along with the RAGAS scores. Section 5.3 evaluates the generation accuracy of Community-Graph RAG when using a local search algorithm. Section 5.4 summarizes the benchmarking results for remaining metrics, including execution time, token usage, CPU load, and memory consumption. Finally, Section 5.5 details the token cost of knowledge graph construction, and Section 5.6 provides a cost analysis for each system.

5.1 Generation Accuracy

This section presents the evaluation results of the answer generation accuracy for the RAG systems: Naive RAG, Graph RAG, and Recursive RAG. Two main analyses were performed. First, the overall accuracy of each system was measured to assess their general performance across all queries. Second, a detailed breakdown of the accuracy per query type was performed to gain insights into the systems' strengths and weaknesses across different categories of questions.

5.1.1 Overall Accuracy

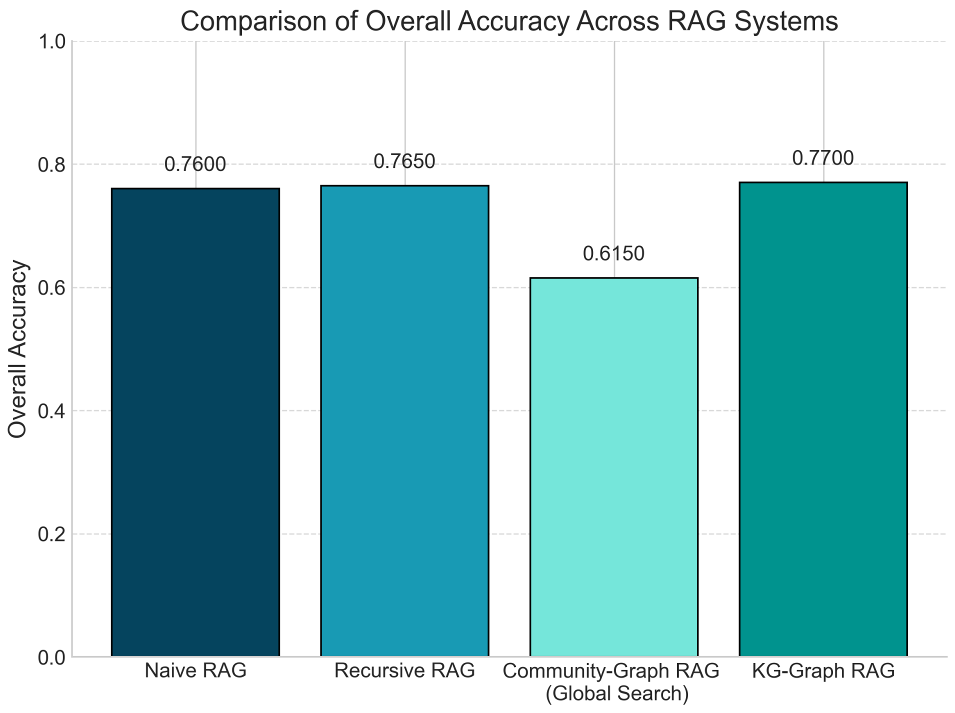


Figure 5.1: Overall answer accuracy for naive RAG, Recursive RAG, Community-Graph RAG (Global Search) and KG-Graph RAG

Figure 5.1 illustrates the overall generation accuracy of the four RAG systems. As shown in the figure, KG-Graph RAG achieved the highest overall accuracy at 77.0%, closely followed by Recursive RAG at 76.5% and Naive RAG at 76.0%. Community-Graph RAG exhibited noticeably lower performance, achieving an accuracy of 61.5%. These results suggest that vector-based RAG systems exhibited comparable or improved performance relative to Graph RAG. Section 5.1.2, which presents a breakdown of accuracy by query type, provides further insight into the factors underlying these overall results. By analyzing each system's performance across different query categories, it becomes possible to better understand the discrepancies in accuracy between the systems.

5.1.2 Accuracy per Query Type

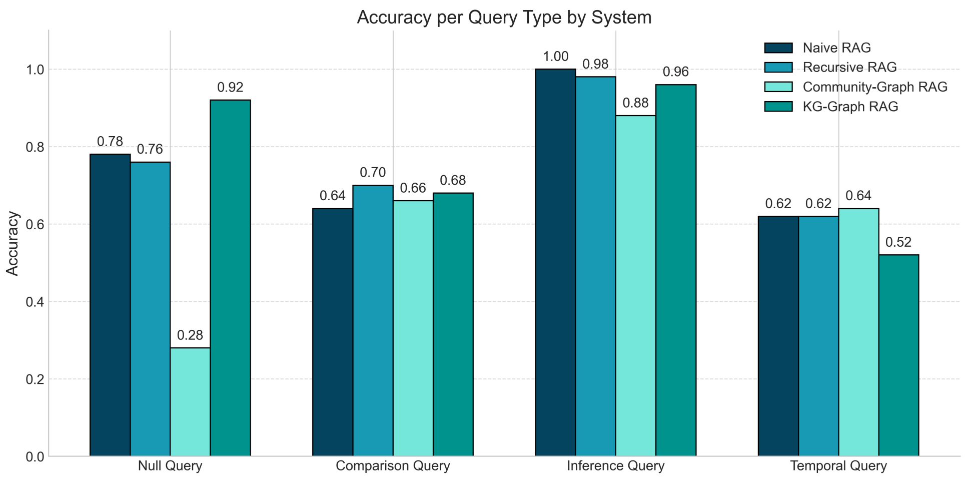


Figure 5.2: Accuracy detailed per query type for naive RAG, Recursive RAG, Community-Graph RAG (Global Search) and KG-Graph RAG

A more detailed analysis of answer accuracy across different query types was performed for each system, as shown in Figure 5.4. For null queries, KG-Graph RAG achieved the highest accuracy at 92.0%. Naive RAG and Recursive RAG followed with similar scores of 78.0% and 76.0%, respectively, while Community-Graph RAG performed considerably worse at 28.0%. In comparison queries, all systems performed similarly, with Recursive RAG achieving the highest accuracy at 70.0%, followed by KG-Graph RAG at 68.0%. For inference queries, Naive RAG achieved perfect accuracy (100.0%), with Recursive RAG and KG-Graph RAG close behind at 98.0% and 96.0%, respectively. Community-Graph RAG lagged behind at 88.0%. Lastly, for temporal queries, Naive RAG and Recursive RAG both achieved an accuracy of 62.0%, while Community-Graph RAG performed slightly better at 64.0%. KG-Graph RAG, in contrast to its strong performance on other query types, recorded the lowest accuracy for temporal queries at 52.0%.

5.2 Retrieval Accuracy

This section presents the results of the retrieval evaluation. First, the retrieved context of each system was evaluated against the gorund-truth context based on semantic similarity using BERTScore. Furthermore, RAGAS, a framework that employs an LLM for evaluation was used to provide an additional dimension in our evaluation. For retrieval evaluation, null queries were excluded, as they do not have an associated context. For KG-Graph RAG, only the retrieved context from the vector-based search was evaluated. The graph-derived context, composed of knowledge triplets, was excluded due to repetition of entity names, which could have artificially inflated BERTScore values.

5.2.1 BERTScore

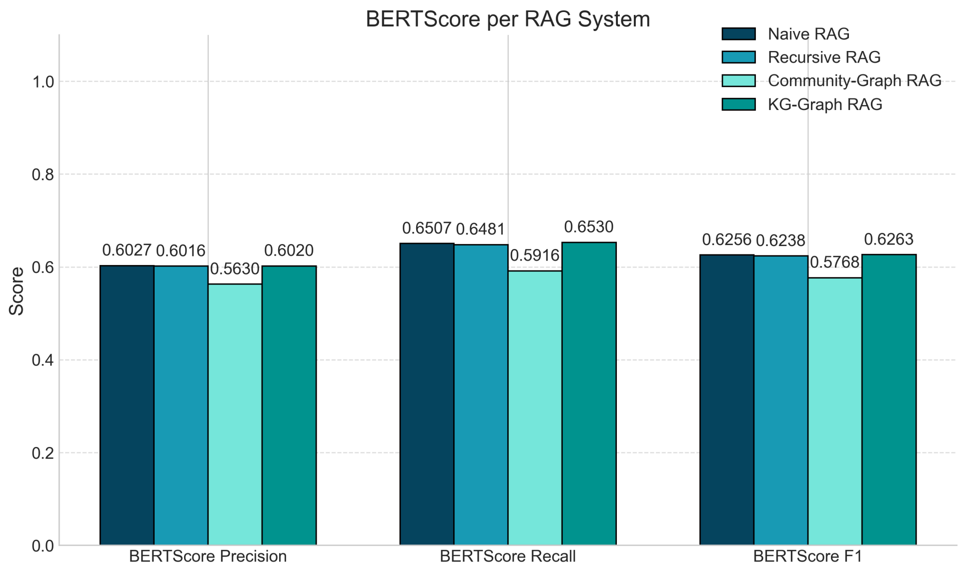


Figure 5.3: Retrieval accuracy per RAG system measured with BERTScore

Overall, Naive RAG, Recursive RAG and KG-Graph RAG had almost identical scores across all BERTScore metrics, which can be explained by the fact that they use semantic similarity-based retrieval. Community-Graph RAG, on the other hand, had lower scores on all BERTScore metrics, with an F1 of 0.5768. This result suggests that although Community-Graph RAG may leverage a rich retrieval structure, it may also introduce noise or irrelevant information that reduces the semantic match of the final retrieved context in comparison to the ground-truth. An additional factor that may have contributed to the lower score of Community-Graph RAG is that the context it retrieves has undergone multiple layers of LLM-based summarization, which may have introduced abstraction or information loss that negatively impacted alignment with the ground-truth.

5.2.2 RAGAS

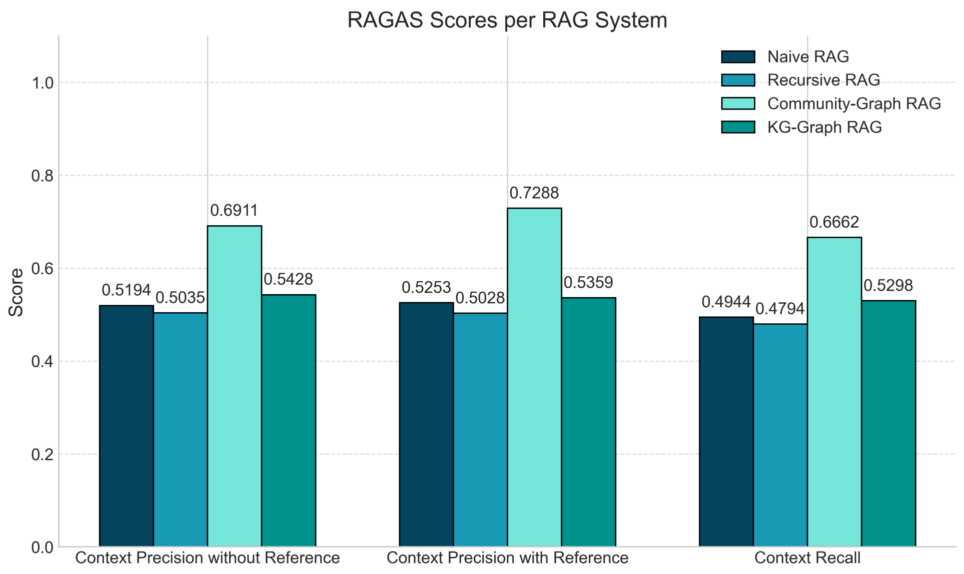


Figure 5.4: RAGAS metrics for each RAG system

To further assess the quality of retrieved context, RAGAS metrics were employed, namely Context Precision without Reference, Context Precision with Reference, and Context Recall. These metrics evaluate how relevant the retrieved context is in supporting the final answer, with and without the ground-truth answer for comparison.

Community-Graph RAG outperformed all other RAG systems according to RAGAS scores, in contrast to the BERTScore metrics.

It achieved the highest values in both context precision and recall (around 70%), indicating that the context retrieved by Community-Graph RAG aligns more closely with the information required to answer the queries, both when evaluated independently and against the ground-truth answers. The remaining three systems received comparable scores, ranging from 49% to 54% across all metrics, with KG-Graph RAG slightly ahead, followed by Naive RAG, and finally Recursive RAG.

5.3 Community-GraphRAG Local Search

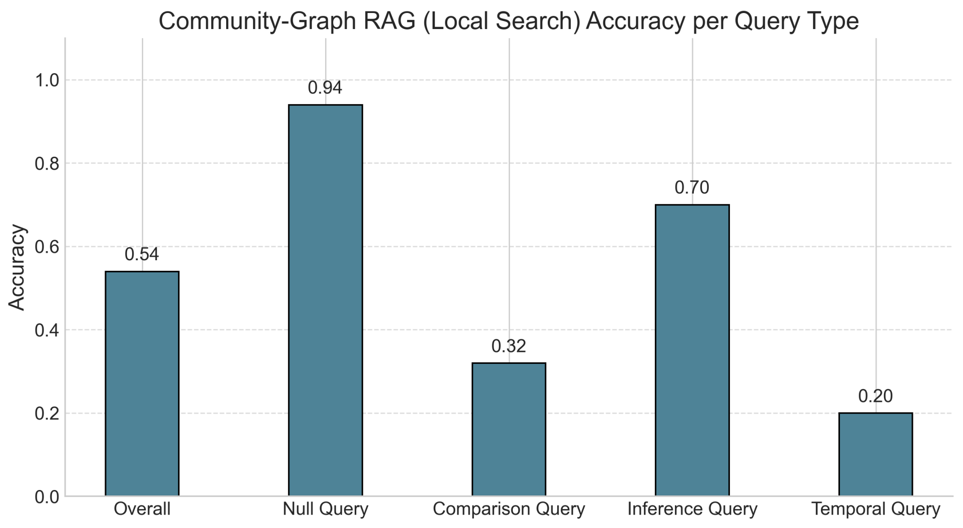


Figure 5.5: Community-Graph RAG accuracy achieved using local search

Figure 5.5 details the accuracy achieved by Community-Graph RAG using the local search algorithm. This experimental analysis was conducted to explore whether local search performs better for specific query types compared to global search. Specifically, our hypothesis was that local search would have better accuracy on null queries, as the search is conducted on lower-level, more detailed communities (and their neighbors), which have summaries that semantically match the user queries.

Local search achieved significantly better accuracy on null queries, at 94%, making it the highest value among all constructed

RAG systems. However, local search performed worse on all other query types, especially temporal queries, with an accuracy of only 20%, bringing the overall accuracy to just 54%. Based on these results, a hybrid search approach that can switch between local and global search depending on the nature of the data or the query type would likely yield more satisfactory results.

5.4 Other Metrics

This section presents the results of the resource consumption metrics, including execution time, token usage, CPU utilization, and memory consumption.

5.4.1 Query Execution Time

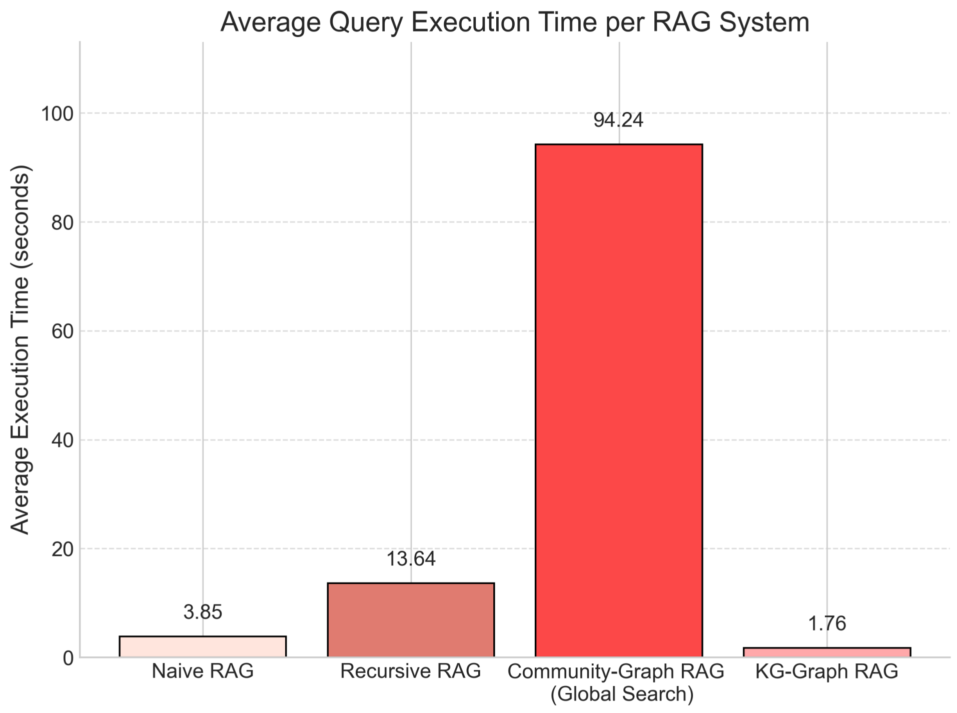
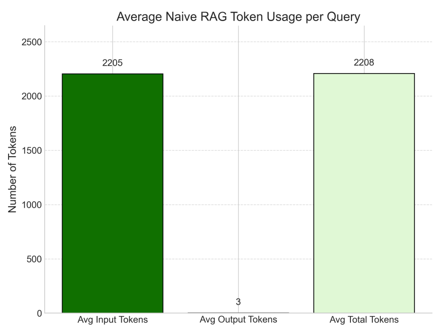


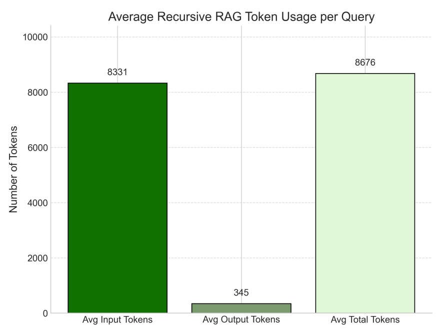
Figure 5.6: Average execution time per query for all RAG systems

Figure 5.6 illustrates the average execution time per query for each of the implemented systems. KG-Graph RAG demonstrated the lowest response latency, averaging 1.76 seconds per query. Naive RAG followed with an average response time of 3.85 seconds, while Recursive RAG exhibited a higher latency of 13.64 seconds. The system with the longest execution time was Community-Graph RAG, averaging 94.24 seconds per query. This substantial delay was primarily due to OpenAI's rate limiting, which caused repeated interruptions during query generation. As many API requests per query were blocked, the system had to pause and retry after several seconds, significantly increasing the overall execution time.

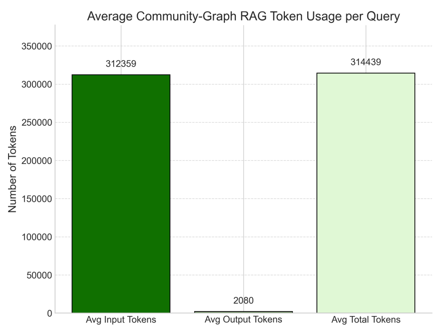
5.4.2 Token Usage



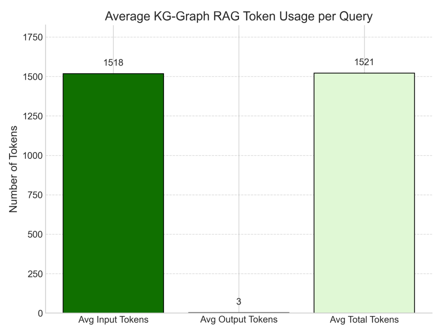
(a) Naive RAG



(b) Recursive RAG



(c) Community-Graph RAG



(d) KG-Graph RAG

Figure 5.7: Average token usage per query for each RAG system

As observed in Figure 5.7, Community-Graph RAG had significantly higher token usage compared to all other RAG systems, with an average of 314,439 tokens per query. The majority of these tokens are input tokens, which aligns with the number of LLM calls made to refine the community summaries. This result indicates that

Community-Graph RAG is more resource-intensive in terms of token consumption, due to its process of iterative summary refinement. In contrast, KG-Graph RAG had the lowest token usage, with an average of 1,521 tokens per query. Naive RAG followed with a total token usage of 2,208 tokens, while Recursive RAG had a higher token usage of 8,676 tokens. The latter's increased token usage can be attributed to the recursive nature of the system, where previous context is passed to the model in each iteration.

5.4.3 Hardware Utilization

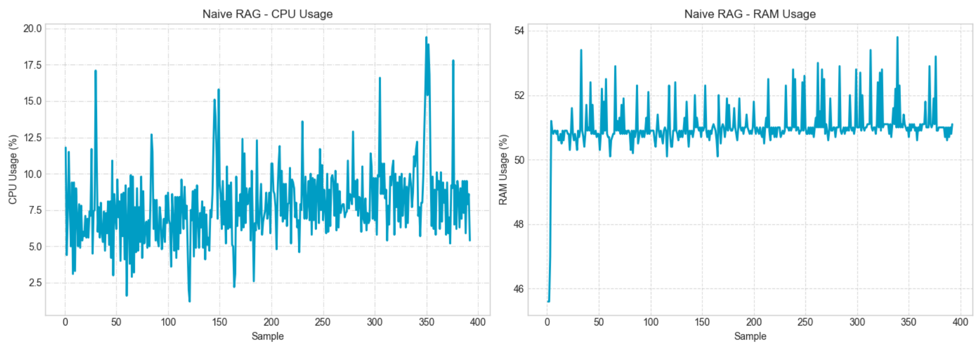


Figure 5.8: CPU and memory usage measured every 2 seconds for Naive RAG

As shown in Figure 5.8, the CPU usage for Naive RAG fluctuates between 2% and 12%, with occasional spikes reaching almost 20%. In contrast, memory usage remains relatively stable, ranging from approximately 50% to 54%. Overall, both CPU and memory usage appear to be moderate throughout the evaluation.

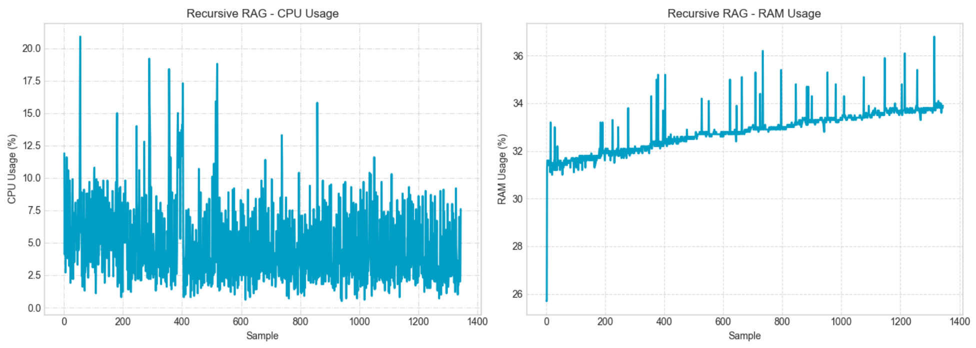


Figure 5.9: CPU and memory usage measured every 2 seconds for Recursive RAG

Figure 5.9 illustrates the CPU and RAM usage for the Recursive RAG system. Similar to Naive RAG, CPU usage fluctuates between 2% and 10%, with a greater concentration of values near the lower bound compared to the Naive RAG system. The RAM usage, however, exhibits a slight upward trend over the observed samples, starting at around 31% and gradually increasing to approximately 34% by the end. This suggests that the memory footprint of the Recursive RAG system increases as it processes more data or performs additional recursive retrieval steps.

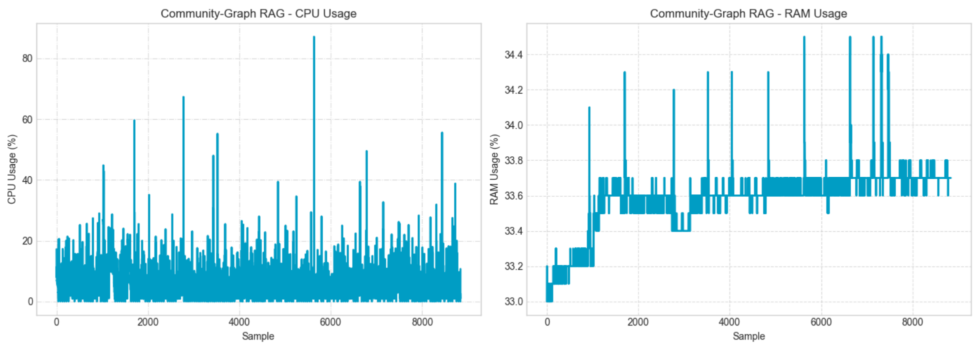


Figure 5.10: CPU and memory usage measured every 2 seconds for Community-Graph RAG

Figure 5.10 shows the CPU and RAM usage for the Community-Graph RAG system. The CPU usage ranges from 0% to 20%, with a higher concentration of values near 0% compared to the other systems. However, there are occasional spikes reaching up to 80%, which are likely caused by graph traversal operations. In contrast, lower CPU usage is typically observed when the system is primarily making OpenAI API calls. The memory usage remained consistent between 33% to 34%.

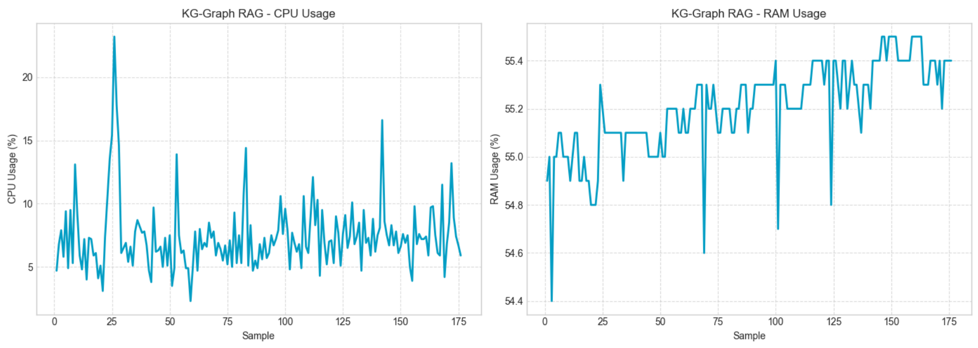


Figure 5.11: CPU and memory usage measured every 2 seconds for KG-Graph RAG

Figure 5.11 shows the CPU and RAM usage for the KG-Graph RAG. The system showed stable CPU usage ranging between 5% and 10%. Similarly, it showed stable RAM usage between 54% and 55%.

Overall, all four systems demonstrated moderate and manageable resource usage, with some variation in usage patterns. KG-Graph RAG showed the most consistent performance, while Community-Graph RAG exhibited occasional CPU spikes, making it more resource-sensitive during specific operations. These results suggest that all systems operate efficiently within reasonable hardware constraints.

5.5 Knowledge Graph Construction Cost

Table 5.1: Token usage statistics for knowledge graph construction

RAG System	**Input Tokens**	**Output Tokens**	**Total Tokens**
KG-Graph RAG	2,178,000	620,508	2,798,508
Community-Graph RAG	8,496,200	1,768,000	10,264,200

Table 5.1 presents the token usage required to construct the knowledge graphs for KG-Graph RAG and Community-Graph RAG. Community-Graph RAG exhibited significantly higher token

consumption, approximately 10 million tokens in total, compared to around 3 million for KG-Graph RAG. This cost was incurred while building the knowledge graphs from a corpus of only 157 documents, which is relatively small. In real-world applications, the document corpus is likely to be substantially larger and continuously growing.

Given the observed accuracy of Community-Graph RAG, its higher tendency to hallucinate answers, and its substantial token usage per query, the upfront cost may be difficult to justify, especially in scenarios that demand fact-based answer generation rather than broad summaries of the entire knowledge base. The latter use case is more appropriate for Community-Graph RAG, as noted in the Microsoft Graph RAG paper.

KG-Graph RAG, on the other hand, demonstrated both higher accuracy and a lower hallucination rate than all other systems, while incurring fewer tokens for both graph construction and answer generation. This suggests strong potential for production environments, particularly where the knowledge graph can be further optimized (for example, by defining a focused set of entities to extract based on the dataset). Such refinements could yield even higher accuracy at a significantly reduced cost compared to Community-Graph RAG.

The vector-based systems required no token consumption for initial construction and still achieved accuracy levels comparable to KG-Graph RAG.

5.6 Cost Analysis

Table 5.2: Average cost associated with each RAG system, using OpenAI's GPT4o mini

RAG System	**Graph Construction Cost**	**Average Cost per Query**
Naive RAG	-	$0.0003
Recursive RAG	-	$0.0015
KG-Graph RAG	$0.6990	$0.0002
Community-Graph RAG	$2.3352	$0.0481

In this section, we provide an analysis of the cost associated with each of the four RAG systems considered in our study. Table 5.2

outlines both the graph construction cost and the average cost per query for each RAG system (using GPT4o mini). Naive RAG had the lowest overall cost, followed by Recursive RAG. KG-Graph RAG had the lowest average cost per query, although it incurred an initial graph construction cost. Community-Graph RAG had the highest overall cost, with a graph construction cost of $2.34 and a query cost of $0.0481, making it the least cost-effective system.

Chapter 6

Discussion

As presented in Chapter 5, KG-Graph RAG emerges as a promising alternative to traditional vector-based RAG systems. Although it slightly underperformed compared to Naive RAG and Recursive RAG on certain query types, its ability to correctly identify null queries and avoid generating hallucinated answers is noteworthy, achieving an accuracy of 92%. Furthermore, KG-Graph RAG exhibited the lowest token usage among all four systems, consistently moderate hardware utilization, and fast query response times.

Notably, KG-Graph RAG demonstrated near-perfect performance on inference-based queries, indicating its strong reasoning capabilities within the structured knowledge graph. However, its performance on temporal queries was the weakest of the systems evaluated. This may suggest that the constructed knowledge graph struggled to preserve temporal relationships within the data. Moreover, the initial cost of constructing the knowledge graph represents a trade-off, although it may be justifiable in contexts where minimizing hallucinations is of high importance.

Community-Graph RAG demonstrated comparable or improved performance on comparison and temporal queries compared to the other systems. This makes it a strong candidate for multi-hop use cases, particularly those involving temporal reasoning. However, the system exhibited a high hallucination rate, achieving an accuracy of only 28%. It also had significantly higher token usage per query, higher query latency, and a greater initial cost for graph construction compared to KG-Graph RAG.

While Community-Graph RAG shows potential for MHQA, these strengths are offset by its overall low accuracy, which was the lowest among all evaluated systems. The system may be better suited for tasks that require longer and more comprehensive answers that summarize information from the entire text corpus. This is its intended use case, as highlighted in the Microsoft paper. For QA tasks, the summarization-based retrieval approach seems to introduce noise, which likely contributed to the high hallucination rate and poor performance on inference queries.

This limitation is also reflected in the BERTScore of the retrieved context, where Community-Graph RAG scored lower than the other systems. Interestingly, RAGAS assigned it a higher score (around 20% higher than all other systems), possibly because it included more context, which was interpreted as more comprehensive. However, in the context of question answering, this additional information appears to have misled the LLM, resulting in reduced accuracy.

One approach to improving Community-Graph RAG's accuracy is to use a hybrid search strategy, where local search is applied to null queries. Local search significantly reduced hallucinations compared to global search, resulting in an overall performance that is more comparable to the other evaluated systems.

Naive RAG and Recursive RAG, which performed similarly, offered a reasonable trade-off between accuracy and cost. However, both systems exhibited a higher hallucination rate compared to KG-Graph RAG, which may be a critical factor in applications where factual correctness is essential. Notably, both systems achieved perfect or near-perfect accuracy on fact-based inference queries. Additionally, they demonstrated low token usage (with Naive RAG using nearly four times fewer tokens), along with moderate latency, CPU, and memory consumption. These characteristics make the vector-based RAG systems strong candidates for general-purpose chatbots, where cost, ease of system construction, efficiency, and responsiveness are prioritized over absolute factual precision.

The results presented in this thesis align with previous research evaluating Naive RAG and Graph RAG. Prior work found that Naive RAG excelled in fact-based retrieval, while Community-Graph RAG demonstrated a greater tendency to hallucinate but achieved higher accuracy in multi-hop retrieval. Additionally, it was similarly reported that KG-Graph RAG performed well on null queries [56]. This thesis extends these findings by offering a detailed analysis of resource consumption, graph construction costs, and a more nuanced evaluation of retrieval quality.

Chapter 7

Conclusion and Future Work

This chapter summarizes the key findings of the thesis, discusses its limitations, and outlines directions for future work and broader reflections on the implications of the research.

7.1 Conclusion

This thesis presented a comparative evaluation of vector-based and knowledge-graph-based RAG systems. Specifically, four RAG systems were constructed: Naive RAG, Recursive RAG, KG-Graph RAG, and Community-Graph RAG. These systems were evaluated on an MHQA dataset. The benchmark assessed answer accuracy, retrieval quality, and resource consumption for each system. Based on the results, KG-Graph RAG emerged as a strong candidate due to its high accuracy, low hallucination rate, and efficient resource usage.

Although Naive RAG and Recursive RAG demonstrated higher hallucination rates, they were still promising alternatives. These systems were easy to construct, incurred no initial knowledge base construction cost, and had moderate token usage, hardware utilization, and response times. This makes them a favorable choice for building general-purpose chatbots.

None of the evaluated systems achieved an overall accuracy above 80 percent on the multi-hop queries. However, there is clear potential for improving KG-Graph RAG to increase its accuracy. For example, based on the specific dataset or use case, entities and relationships could be more selectively defined and extracted during knowledge graph construction. This could make it easier to capture key information. Furthermore, a hybrid system that dynamically switches between retrieval approaches based on the nature of the query or data could leverage the strengths of each method, combining them into a single, more robust system.

7.2 Limitations

Despite the contributions of this thesis, several limitations should be acknowledged.

First, due to economic constraints, the evaluation was conducted on a relatively small dataset consisting of a limited number of multi-hop questions and a small corpus for constructing the knowledge base. In addition, the dataset was restricted to yes/no questions and single-entity answers. A larger and more diverse dataset, including queries that require longer and more elaborate answers, would likely offer a more comprehensive assessment of system performance.

Second, all evaluations were carried out using GPT-4o mini as the sole LLM backend. While this allowed for consistent comparisons, the results may vary when using other models with different reasoning capabilities, token limits, or response behaviors. Exploring alternative or more powerful LLMs, especially for the knowledge graph construction phase, could improve accuracy and yield interesting results.

Third, Community-Graph RAG showed promising results on multi-hop and temporal queries, but it was only tested with a single prompt configuration and minimal parameter tuning. Further experimentation with prompt design, chunking strategy, and recursive depth could improve its performance and reduce hallucinations. The potential of this system was not fully explored within the scope of this thesis.

Finally, the study focused on accuracy and resource consumption metrics but did not evaluate user-facing factors such as latency under high load or robustness to ambiguous or adversarial inputs. These factors are crucial for real-world deployment and should be explored further.

7.3 Future Work

To improve the accuracy of the results and strengthen the findings of this study, future work should consider using a larger dataset paired with a corresponding text corpus. This would enable more comprehensive evaluation of the generated answers using metrics such as faithfulness and answer relevance.

Another promising direction for future research is to focus on enhancing the performance of KG-Graph RAG. For example, experiments could be conducted to refine the prompt engineering techniques used to generate the knowledge graph. These prompts could be tailored to extract specific entities and types of information based on the task requirements and desired optimization goals. Temporal queries, in particular, could benefit from prompting the language model to pay closer attention to the sequencing and timing of events, thereby improving its ability to reason over temporally grounded information.

Lastly, hybrid systems that combine a Graph RAG implementation with a vector-based RAG approach may be well-suited for certain use cases. Since Naive RAG excels at fact-based retrieval, it could be used for fact-based inference queries. In contrast, Graph RAG can be optimized for more complex, multi-hop reasoning tasks. To support this, queries should first be classified into specific types using a language model, and based on the classification, the appropriate retrieval system would be used to generate the answer.

7.4 Reflections

This thesis project has several relevant economic, environmental and ethical implications.

From an economic perspective, RAG systems can reduce computational costs by limiting the need for large-scale fine-tuning, and instead relying on models augmented with external information. This can be particularly beneficial for organizations with limited computational resources, lowering the barrier to adopting LLM solutions. Moreover, this thesis highlights the trade-off between accuracy and cost across different RAG implementations, offering insights that support informed, cost-effective decisions when selecting a suitable RAG strategy.

The environmental impact of LLMs is an increasingly important concern. While RAG systems are more efficient than training massive models from scratch or performing frequent fine-tuning, they can still consume substantial energy with each query. However, this thesis has helped shed light on the average cost per query across different systems, offering a better understanding of which approaches may be less sustainable and should potentially be avoided in resource-constrained or environmentally sensitive contexts.

From an ethical standpoint, it is crucial to ensure that the LLM system does not produce hallucinated or false information. For this reason, this study has evaluated each implemented RAG system's ability to accurately detect queries that are not supported by the external knowledge source. In fields such as healthcare, finance, or law, RAG systems with low hallucination rates are essential. They can enhance efficiency without compromising the reliability and trustworthiness of the information provided.

References

[1] D. Edge *et al.*, "From local to global: a graph rag approach to query-focused summarization", *arXiv preprint arXiv:2404.16130*, 2024.

[2] OpenAI, *Gpt-4o mini: advancing cost-efficient intelligence*, Accessed: 2025-05-17, 2024. [Online]. Available: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[3] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges", *Multimedia tools and applications*, vol. 82, no. 3, pp. 3713–3744, 2023.

[4] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning", *arXiv preprint arXiv:1605.05101*, 2016.

[5] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model", *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, 2020.

[6] J. Devlin, "Bert: pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv: 1810.04805*, 2018.

[7] A. Vaswani, "Attention is all you need", *Advances in Neural Information Processing Systems*, 2017.

[8] L. Qin *et al.*, "Large language models meet nlp: a survey", *arXiv preprint arXiv:2405.12819*, 2024.

[9] T. B. Brown *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.14165.

[10] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, “Parameter-efficient fine-tuning for large models: a comprehensive survey”, *arXiv preprint arXiv:2403.14608*, 2024.

[11] K. M. Tarwani and S. Edem, “Survey on recurrent neural network in natural language processing”, *Int. J. Eng. Trends Technol*, vol. 48, no. 6, pp. 301–304, 2017.

[12] W. Fan *et al.*, “A survey on rag meeting llms: towards retrieval-augmented large language models”, in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 6491–6501.

[13] Z. Ji *et al.*, “Survey of hallucination in natural language generation”, *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[14] Z. Ji *et al.*, “Rho (ρ): reducing hallucination in open-domain dialogues with knowledge grounding”, *arXiv preprint arXiv:2212.01588*, 2022.

[15] S. Wu *et al.*, “Retrieval-augmented generation for natural language processing: a survey”, *arXiv preprint arXiv:2407.13193*, 2024.

[16] J. Wu *et al.*, “Medical graph rag: towards safe medical large language model via graph retrieval-augmented generation”, *arXiv preprint arXiv:2408.04187*, 2024.

[17] C. Ryu *et al.*, “Retrieval-based evaluation for llms: a case study in korean legal qa”, in *Proceedings of the Natural Legal Language Processing Workshop 2023*, 2023, pp. 132–137.

[18] M. Arslan, S. Munawar, and C. Cruz, “Business insights using rag–llms: a review and case study”, *Journal of Decision Systems*, pp. 1–30, 2024.

[19] J. Achiam *et al.*, “Gpt-4 technical report”, *arXiv preprint arXiv:2303.08774*, 2023.

[20] A. Q. Jiang *et al.*, “Mistral 7b”, *arXiv preprint arXiv:2310.06825*, 2023.

[21] S. Borgeaud *et al.*, “Improving language models by retrieving from trillions of tokens”, in *International conference on machine learning*, PMLR, 2022, pp. 2206–2240.

[22] M. Douze *et al.*, *The faiss library*, 2025. arXiv: 2401.08281 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2401.08281.

[23] Pinecone Systems, Inc., *Pinecone: the vector database to build knowledgeable ai*, https://www.pinecone.io/, Accessed: 2025-04-23, 2025.

[24] Chroma, *Chroma is the open-source ai application database. batteries included.* https://www.trychroma.com/, Accessed: 2025-04-23, 2025.

[25] LangChain Contributors. “Langchain: a framework for developing applications powered by language models”. Accessed: 2025-05-15, LangChain. (2022), [Online]. Available: https://github.com/langchain-ai/langchain.

[26] Y. Gao *et al.*, “Retrieval-augmented generation for large language models: a survey”, *arXiv preprint arXiv:2312.10997*, 2023.

[27] T. H. Haveliwala, A. Gionis, D. Klein, and P. Indyk, “Evaluating strategies for similarity search on the web”, in *Proceedings of the 11th international conference on World Wide Web*, 2002, pp. 432–442.

[28] S. Zhao, Y. Huang, J. Song, Z. Wang, C. Wan, and L. Ma, “Towards understanding retrieval accuracy and prompt quality in rag systems”, *arXiv preprint arXiv:2411.19463*, 2024.

[29] X. Wang *et al.*, “Searching for best practices in retrieval-augmented generation”, in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024, pp. 17 716–17 736.

[30] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan, “Query rewriting for retrieval-augmented large language models”, *arXiv preprint arXiv:2305.14283*, 2023.

[31] H. Koo, M. Kim, and S. J. Hwang, “Optimizing query generation for enhanced document retrieval in rag”, *arXiv preprint arXiv:2407.12325*, 2024.

[32] C.-M. Chan *et al.*, “Rq-rag: learning to refine queries for retrieval augmented generation”, *arXiv preprint arXiv: 2404.00610*, 2024.

[33] L. Gao, X. Ma, J. Lin, and J. Callan, “Precise zero-shot dense retrieval without relevance labels”, *arXiv preprint arXiv:2212.10496*, 2022.

[34] Y. Yu *et al.*, “Rankrag: unifying context ranking with retrieval-augmented generation in llms”, *Advances in Neural Information Processing Systems*, vol. 37, pp. 121 156–121 184, 2025.

[35] Y. Gao, Y. Xiong, M. Wang, and H. Wang, “Modular rag: transforming rag systems into lego-like reconfigurable frameworks”, *arXiv preprint arXiv:2407.21059*, 2024.

[36] J. Wei *et al.*, “Chain-of-thought prompting elicits reasoning in large language models”, *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[37] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal, “Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions”, *arXiv preprint arXiv:2212.10509*, 2022.

[38] L. Wang, H. Chen, N. Yang, X. Huang, Z. Dou, and F. Wei, “Chain-of-retrieval augmented generation”, *arXiv preprint arXiv:2501.14342*, 2025.

[39] T. Liu *et al.*, “Statistical rejection sampling improves preference optimization”, *arXiv preprint arXiv:2309.06657*, 2023.

[40] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, “Raptor: recursive abstractive processing for tree-organized retrieval”, *arXiv preprint arXiv:2401.18059*, 2024.

[41] G. Kim, S. Kim, B. Jeon, J. Park, and J. Kang, “Tree of clarifications: answering ambiguous questions with retrieval-augmented large language models”, *arXiv preprint arXiv: 2310.14696*, 2023.

[42] Neo4j, *Neo4j: graphs for everyone*, Accessed: 2025-05-17, 2025. [Online]. Available: https://github.com/neo4j/neo4j.

[43] Y. Hu, Z. Lei, Z. Zhang, B. Pan, C. Ling, and L. Zhao, “Grag: graph retrieval-augmented generation”, *arXiv preprint arXiv: 2405.16506*, 2024.

[44] D. Sanmartin, “Kg-rag: bridging the gap between knowledge and creativity”, *arXiv preprint arXiv:2405.12035*, 2024.

[45] X. Zhu, Y. Xie, Y. Liu, Y. Li, and W. Hu, *Knowledge graph-guided retrieval augmented generation*, 2025. arXiv: 2502.06864 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2502.06864.

[46] R. Yang, H. Xue, I. Razzak, H. Hacid, and F. D. Salim, *Kg-irag: a knowledge graph-based iterative retrieval-augmented generation framework for temporal reasoning*, 2025. arXiv: 2503.14234 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2503.14234.

[47] V. A. Traag, L. Waltman, and N. J. Van Eck, “From louvain to leiden: guaranteeing well-connected communities”, *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[48] V. Mavi, A. Jangra, A. Jatowt, *et al.*, “Multi-hop question answering”, *Foundations and Trends® in Information Retrieval*, vol. 17, no. 5, pp. 457–586, 2024.

[49] Y. Tang and Y. Yang, “Multihop-rag: benchmarking retrieval-augmented generation for multi-hop queries”, *arXiv preprint arXiv:2401.15391*, 2024.

[50] T. Hu and X.-H. Zhou, “Unveiling llm evaluation focused on metrics: challenges and solutions”, *arXiv preprint arXiv:2404.09135*, 2024.

[51] S. Es, J. James, L. E. Anke, and S. Schockaert, “Ragas: automated evaluation of retrieval augmented generation”, in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 2024, pp. 150–158.

[52] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, “ARES: an automated evaluation framework for retrieval-augmented generation systems”, in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*

(Volume 1: Long Papers), K. Duh, H. Gomez, and S. Bethard, Eds., Mexico City, Mexico: Association for Computational Linguistics, Jun. 2024, pp. 338–354. DOI: 10.18653/v1/2024.naacl-long.20. [Online]. Available: https://aclanthology.org/2024.naacl-long.20/.

[53] Y. Zhou *et al.*, "Trustworthiness in retrieval-augmented generation systems: a survey", *arXiv preprint arXiv:2409.10102*, 2024.

[54] T. Şakar and H. Emekci, "Maximizing rag efficiency: a comparative analysis of rag methods", *Natural Language Processing*, vol. 31, no. 1, pp. 1–25, 2025.

[55] M. Eibich, S. Nagpal, and A. Fred-Ojala, "Aragog: advanced rag output grading", *arXiv preprint arXiv:2404.01037*, 2024.

[56] H. Han *et al.*, *Rag vs. graphrag: a systematic evaluation and key insights*, 2025. arXiv: 2502.11371 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2502.11371.

[57] H. H. Alharahsheh and A. Pius, "A review of key paradigms: positivism vs interpretivism", *Global academic journal of humanities and social sciences*, vol. 2, no. 3, pp. 39–43, 2020.

[58] G. Yepez. "Nano-graphrag". Accessed: 2025-04-11. (2024), [Online]. Available: https://github.com/gusye1234/nano-graphrag.

[59] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, *Bge m3-embedding: multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation*, 2024. arXiv: 2402.03216 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.03216.

[60] Wikipedia contributors. "Iverson bracket". Accessed: 2025-04-16, Wikipedia. (2016), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Iverson_bracket&oldid=714958186.

[61] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, *Bertscore: evaluating text generation with bert*, 2020. arXiv: 1904.09675 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1904.09675.