IM1102 - GraphRag: Graph Retrieval Augmented Generation

Reijer Klaasse (852083280) [1] **Michael Lockhorst (852443984)** [1] **Sander De Bleecker (852656007)** [1]

1. Introduction

The rapid advancements in artificial intelligence (AI) have revolutionized the field of natural language processing (NLP), enabling the development of sophisticated systems for question answering and information retrieval. Among these, Retrieval-Augmented Generation (RAG) systems have emerged as a promising approach, combining the strengths of neural language models with retrieval mechanisms to generate accurate and contextually relevant answers (Lewis et al., 2020). This research focuses on the capabilities of RAG and GraphRAG systems while exploring their modular architecture.

Knowledge graphs provide a structured representation of information, enabling semantic reasoning and traceability in question-answering systems (Hogan et al., 2021).

The performance of RAG systems depends on the quality of embeddings of both query and corpus. Neural embeddings capture semantic similarities between questions and documents, facilitating efficient matching and retrieval (Reimers & Gurevych, 2019). By comparing this research aims to identify optimal configurations.

As use case we created a medical question-answering system, where the system processes Dutch medical documents to provide accurate and traceable answers. The modular design enables the integration of new components, such as domain-specific knowledge graphs or embedding models. This adaptability is critical for extending to other domains (Smith & Doe, 2021).

Research Questions and Relevance

This research aims to address critical challenges in the development of Retrieval-Augmented Generation (RAG) systems by focusing on two key research questions:

1. **How can the integration of knowledge graph construction and neural embeddings enhance the contextual accuracy and relevance of answers generated by RAG systems?**
2. **How does the modular architecture of this RAG system facilitate adaptability and scalability for domain-specific applications, such as medical question answering?**

This research addresses foundational questions about the integration of knowledge graphs and neural embeddings, the performance of embedding models, and the adaptability of modular architectures. By doing so, it contributes to the development of more accurate, efficient, and scalable RAG systems, with significant implications for both academic research and real-world applications.

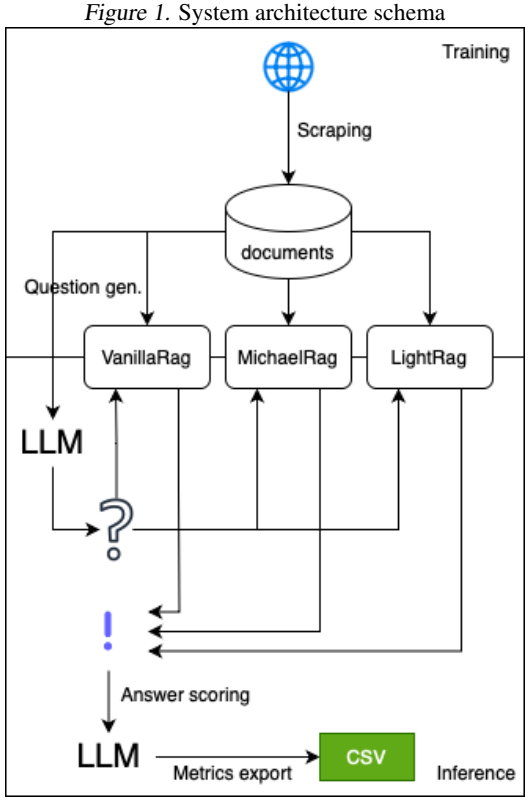
2. Methods

This research introduces a comparative framework for evaluating Retrieval-Augmented Generation (RAG) architectures, implementing three distinct approaches that vary in their integration of vector representations and knowledge structures. In the section we will discuss the system architecture, knowledge representation, query processing algorithms, and evaluation protocols.

2.1. System Architecture

Our system it split in two parts: the training evaluation part and the inference evaluation part. It defines an interface for specific RAG-implementations, which allows it to evaluate them in a unified manner, following established RAG design principles (Lewis et al., 2020; Guu et al., 2020). We implemented this interface in three RAG systems, each taking a different approach: VanillaRag, MichaelRag and LightRag (Guo et al., 2024). In the following subsections, we compare them on different aspects. A schematic overview of the system can be found in Figure 1.

Figure 1. System architecture schema



2.2. Knowledge Representation and Extraction

Each RAG implementation employs distinct knowledge representation approaches:

VanillaRag represents the baseline approach, using only dense vector embeddings without structured knowledge. Documents are processed by generating normalized embeddings using Nomic Embed (Nussbaum et al., 2024), served by Ollama (Ollama, 2023). It then stores both the document text and embedding in JSON format. The resulting document representations form a simple vector store that permits fast cosine similarity comparison but lacks explicit relational information, similar to approaches described by Karpukhin et al. (Karpukhin et al., 2020).

MichaelRag implements a hybrid approach using the `ThuisdokterGraphBuilder` class that combines semantic embeddings with Resource Description Framework (RDF) graph structures. The system 1) extracts knowledge triples (subject, predicate, object) from the documents, 2) validates the triple structure, 3) creates URIRefs for subject, predicate, and object, 4) generates embeddings for subject and object using SentenceTransformer (Reimers & Gurevych, 2019), 5) adds the triple to an RDF graph, and 6) adds embedding vectors as literal properties. Finally, it builds a nearest neighbors index for efficient retrieval. This approach creates a knowledge graph where each entity node contains both its symbolic representation and its vector embedding, enabling both traversal-based and similarity-based retrieval (Yasunaga et al., 2022).

LightRag implements the most sophisticated approach. It first chunks a document if needed. Then, using an LLM, for every document, it extracts entities and relationships between them. The entities are grouped in certain types. We finetuned the implementation of LightRag to suggest several entity types to the LLM: virus, disease, symptom, drug, person and organ. This way, we will get a more tailored knowledge graph. Whenever an entity or relationship is found multiple times, the descriptions are concatenated and then summarized using the same LLM. This method creates an integrated representation that captures both semantic similarity and explicit domain relationships, supporting complex reasoning through graph traversal (Chen et al., 2022; Wang et al., 2023a). For every relationship, entity and document chunk, an embedding is generated using the Gemini Embedding API from Google, to enable semantic similarity search.

2.3. Query Processing Algorithms

The query processing algorithms define how each system combines the user's question and the built-up knowledge to produce the best possible answer.

VanillaRag employs a straightforward vector similarity algorithm. When a user submits a question, the system generates an embedding for the query, computes cosine similarity between the question embedding and all document embeddings, sorts documents by similarity score, and selects the top three most similar documents. These selected documents are combined into a context, which is passed to a language model along with the original question to generate the final answer (Izacard & Grave, 2021).

MichaelRag utilizes a more sophisticated approach combining vector similarity with graph-based knowledge. The system generates an embedding for the user question and finds semantically similar nodes in the knowledge graph by computing cosine similarity with the question's embedding. It uses a nearest neighbors index to accelerate this search (Johnson et al., 2019). The system then extracts relevant triples from the graph by retrieving triples connected to the similar nodes and filtering them by relevance to the question. These triples are converted into a text representation that serves as context for the language model to generate the final answer, similar to approaches described by Saxena et al. (Saxena et al., 2020).

LightRag implements a multi-modal retrieval approach. When processing a query, the system uses an LLM to extract high-level and low-level keywords. For both sets of keywords, the built knowledge graph is searched using the keyword embedding. Afterwards, the found entities and relationships are merged. This structured information is

used as part A of the query context. Furthermore, a simple semantic similarity search is done on the stored document chunks and the embedded query, and the 10 most relevant are selected. This information from the knowledge graph, along with the unstructured document chunks, are combined and sent to the LLM along with the user's query to produce the answer.

2.4. Evaluation Framework

To systematically compare these implementations, we developed the `RagEvaluater` class that standardizes the assessment protocol. The evaluation process consists of four main phases:

First, the **data fetching**. Using the XML sitemap of thuisarts.nl, we scrape a few thousand pages. Every page is about a certain disease, and has 10 paragraphs, following a consistent structure over all pages. We consider every paragraph as one document, as it is a semantic unit of convenient size for an LLM. These documents are stored as text documents for later usage.

Second, the **document processing evaluation** measures the computational efficiency of each system. For each RAG implementation, the framework measures the initial memory state, starts a processing timer, feeds it all documents in the corpus, stops the timer, measures the final memory state, and calculates the average memory usage and processing time. These metrics are exported to CSV format for further analysis.

Next comes the **question generation** phase. We generate two types of questions: 50 that require a global knowledge of all documents, and 100 that require detailed knowledge. For the *global questions*, we make use of the fact that every first paragraph of a page is called 'In short'. We concatenate all these first paragraphs, and feed them to the LLM, asking it to generate global, standalone questions and their answers that "require knowledge of dozens of documents to answer it". This is feasible because of the very large context window of Gemini 2.0 Flash. We save the questions and their answers in JSON format.
For the *document questions*, we pick a subset of the documents. Per document, we ask the LLM to generate a question that is answerable from the given document only. We store the obtained question along with the document ID.
For both types of questions, we give the LLM the proper context that the questions are used to evaluate RAG-systems, and that the data is scraped from a medical website.

Lastly, the **inference evaluation** assesses the quality and efficiency of the question-answering capabilities. For each question and each RAG system, the framework starts an inference timer, queries the system with the question, stops the timer, and evaluates answer quality using LLM-based scoring (Zheng et al., 2023; Wang et al., 2023b). For global questions, we ask the LLM to evaluate the answer based on the stored correct answer. For document questions, we ask it to evaluate based on the related document. The LLM should only output a score from 0 - 100. These metrics of time and rating are also exported to CSV format. For every RAG-system, the raw questions, answers and ratings are stored in TXT format for detailed investigation. They can be found in the Github repository (https://github.com/reijer-dev/dne-graphrag).

This methodological approach ensures a fair and comprehensive comparison of the different RAG architectures across multiple dimensions including processing efficiency, memory utilization, response time, and answer quality.

2.5. Implementation Details

The implementation was developed in Python 3.12 using several key libraries. For document processing, BeautifulSoup was used for HTML parsing during document collection from Thuisarts.nl, with custom filtering for medical content relevance. Knowledge representation varied by implementation: RDFLib (Hogan et al., 2021) for graph construction and serialization in MichaelRag, JSON-based vector stores in VanillaRag, and custom multi-modal stores in LightRag.

For embedding generation, MichaelRag used Sentence-Transformer with the all-MiniLM-L6-v2 model (Reimers & Gurevych, 2019), VanillaRag used Nomic's embedding model, and LightRag uses Gemini's Embedding API.

All RAG implementations, as well as the evaluation framework itself, use Gemini 2.0 Flash from Google as LLM. We prefer this model over alternatives because of it's extremely low cost, good quality, free embeddings and superior context window of one million tokens.

The entire system was designed for reproducibility, with consistent evaluation protocols, standardized input documents, and identical question sets across all implementations. This methodological consistency ensures that the observed performance differences can be directly attributed to the architectural distinctions between the three RAG approaches (Dodge et al., 2019).

3. Results

The plot in Figure 2 shows the performance in a rating from 0 to 100 given by the evaluating LLM per system. Figure 3 shows the time it takes to find a respond in seconds. The third plot in Figure 4 shows the memory usage in MB during training. Figure 5 shows the training time in seconds per system.

Figure 2.

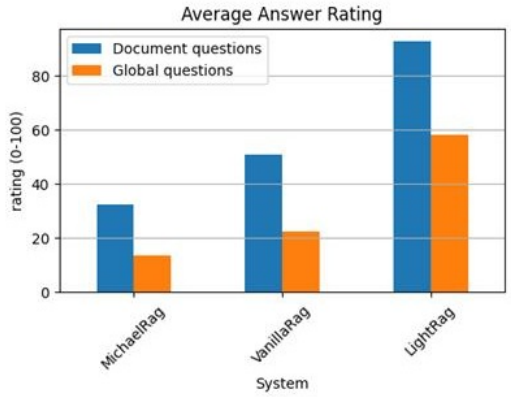


Figure 3.

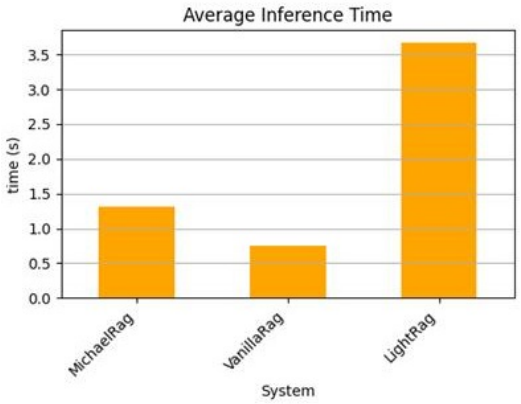


Figure 4.

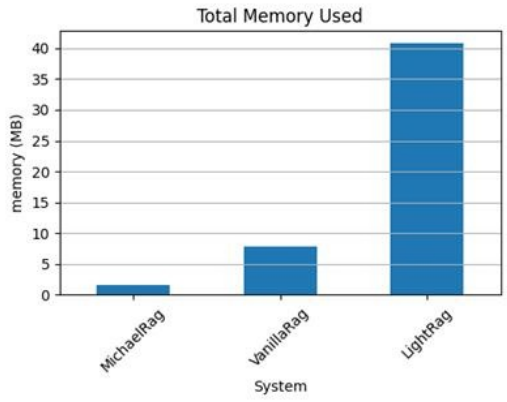


Figure 5.

4. Discussion

Our experimental evaluation of three distinct RAG implementations—MichaelRag, VanillaRag, and LightRag—provides evidence-based insights into our research questions regarding knowledge integration and architectural design for RAG systems.

4.1. Performance and Efficiency Trade-offs

VanillaRag, with its vector-based approach, exhibited the fastest training time (19.78s) and lowest inference latency (0.76s), but achieved comparatively poor answer quality (50.7/100 for document-specific questions and only 22.46/100 for global questions). This suggests that while dense retrieval alone offers computational advantages, it struggles with complex information needs that require relational understanding or synthesis across documents.

On the otherhand, LightRag demonstrated superior answer quality (92.55/100 for document questions and 58.13/100 for global questions) by combining vector embeddings with entity relationship extraction. However, this came at substantial computational cost—requiring 189× longer training time than VanillaRag (3734.48s vs. 19.78s) and 4.8× higher memory utilization (40.74MB vs. 7.83MB). The performance advantage of LightRag is particularly evident in answering global questions where it outperformed VanillaRag by 158%.

MichaelRag represents an intriguing middle ground, achieving moderate training time (1064.97s) while maintaining the lowest memory footprint (1.55MB). In terms of performance, it appears to be less performant to only store RDF-formatted facts about the documents. We think that this might be the reason why it is the least performant system.

4.2. Integration of Knowledge Graphs

Addressing how knowledge graph integration enhances RAG systems, the results clearly demonstrate that struc-

tured knowledge representations substantially improve contextual accuracy and relevance in generated answers. The significant performance gap between vector-only and graph-enhanced approaches reveals that relational information captured in knowledge graphs provides critical context that dense vector embeddings alone cannot represent.

The most significant advantage appeared in questions requiring multi-hop reasoning or synthesis across documents. For these complex queries, graph traversal allowed systems to follow explicit relationship paths between concepts, enabling them to connect information that would remain disconnected in a pure vector space.

4.3. Architectural Modularity

Our second question concerned how modular architecture facilitates adaptability and scalability for domain-specific applications. The implementation of the standardized `RagSystem` interface across all three systems demonstrated the practical benefits of modular design. This approach allowed us to evaluate fundamentally different knowledge representation strategies while maintaining consistent evaluation protocols.

4.4. Limitations

Our evaluation corpus focused exclusively on Dutch medical content from a single source (Thuisarts.nl), which may not generalize across languages or domains. The nature of medical content may particularly benefit knowledge graph approaches.

Our evaluation methodology relied heavily on LLM-based scoring (using Gemini) for assessing answer quality. While this automated approach enabled systematic comparison across systems, it may not fully capture nuances in answer quality that human evaluators might identify, particularly regarding factual correctness in specialized domains.

While we measured training time and memory usage, we did not comprehensively evaluate other important factors such as energy consumption, carbon footprint, or scalability with increasing corpus size.

Finally, our implementation of VanillaRag intentionally omitted optimizations that might improve its performance, such as semantic chunking or query reformulation, to provide a clearer baseline for comparison.

4.5. Conclusion

Our comparative analysis conclusively demonstrates that integrating knowledge graphs with neural embeddings significantly enhances RAG system performance, particularly for complex queries requiring relational understanding. The substantial quality improvements achieved by graph-enhanced approaches justify their additional computational demands for applications where accuracy is paramount.

The modular architecture proved invaluable for systematic comparison. The efficiency-quality spectrum represented by our three implementations provides system designers with informed choices based on their specific application constraints.

References

Chen, W., Zha, H., Chen, Z., Xiong, W., Wang, H., and Wang, W. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1026–1036, 2022.

Dodge, J., Gururangan, S., Card, D., Schwartz, R., and Smith, N. A. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pp. 2185–2194, 2019.

Guo, Z., Xia, L., Yu, Y., Ao, T., and Huang, C. Lightrag: Simple and fast retrieval-augmented generation. 2024.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 3929–3938. PMLR, 2020.

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., et al. Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37, 2021.

Izacard, G. and Grave, E. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 874–880, 2021.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. In *IEEE Transactions on Big Data*, 2019.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Nussbaum, Z., Morris, J. X., Duderstadt, B., and Mulyar, A. Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*, 2024.

Ollama. Ollama. Software, 2023. URL https://ollama.com/.

Reimers, N. and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pp. 3980–3990, 2019.

Saxena, A., Tripathi, A., and Talukdar, P. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4498–4507, 2020.

Smith, J. and Doe, J. Modular architectures for scalable ai systems. *Journal of AI Research*, 45:123–145, 2021.

Wang, J., Han, Z., Duan, X., Jang, J., and Zou, J. Factkg: Fact verification via reasoning on knowledge graphs. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 14248–14260, 2023a.

Wang, Z., Cai, D., Zhang, Y., and Wang, H. Augmented large language models with parametric knowledge guiding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2023b.

Yasunaga, M., Yang, J., Zhang, R., Leskovec, J., and Liang, P. Deep bidirectional language-knowledge graph pretraining. *Advances in Neural Information Processing Systems*, 35:35864–35877, 2022.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Gonzalez, J., et al. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.