THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Data Science Program

Capstone Report - Fall 2024

MyRAG - Advancing Retrieval-Augmented Generation with Comparative Analysis of Standard RAG, ColBERT Reranking, and RAPTOR Architectures

Meet Daxini

supervised by
Amir Jafari

Abstract

Retrieval-Augmented Generation (RAG) has emerged as a powerful paradigm for providing personalized, relevant, and current information to user queries by combining large language models (LLMs) with external retrieval modules. This paper presents MyRAG, an open-source system that unifies and compares different state-of-the-art RAG architectures and retrieval techniques. This paper explores embeddings, vector databases, and advanced retrieval methods, including ColBERT-based reranking and the hierarchical summarization-based RAPTOR architecture. By evaluating different datasets, it demonstrates how different configurations impact retrieval accuracy and downstream QA performance. The results offer insights into optimizing RAG pipelines, guiding both practitioners and researchers toward more efficient and effective retrieval-augmented generation.

Contents

1 Introduction

The exponential growth of digital information has increased the demand for intelligent systems that can efficiently retrieve relevant context and answer complex questions. Large Language Models (LLMs) often possess extensive parametric knowledge, but may lack reliable, up-to-date information. Retrieval-Augmented Generation (RAG) has gained prominence as a solution to this challenge, bridging large-scale language understanding with external retrieval components to produce more grounded and accurate responses [1, 2].

In RAG pipelines, the LLM accesses external knowledge sources, retrieving relevant documents or chunks to augment its prompt. This approach enhances the model's factual accuracy, reduces hallucinations, and updates knowledge without retraining the entire model. As the sizes of the context window continue to expand [3], it is increasingly practical to provide LLMs with larger and more diverse sets of the retrieved context.

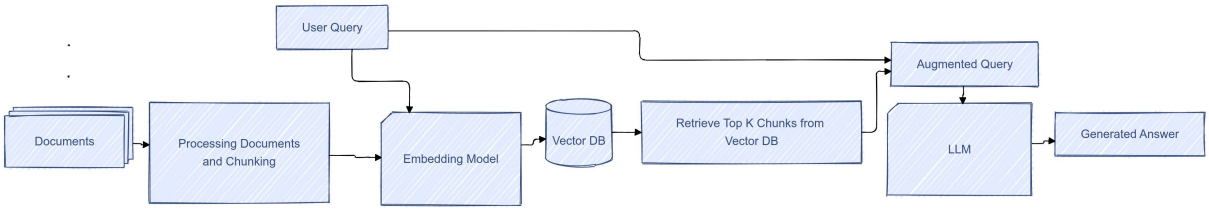

Figure 1: Flow chart illustrating the standard RAG pipeline.

Yet, not all retrieval pipelines are created equal. Various embedding models, databases (like graph, vector, etc), and advanced retrieval techniques can be combined to form a RAG system. For instance, This paper explores How Standard Rag with ColBERT [4] re-ranking can refine retrieved documents, while the RAPTOR architecture [5, 6] organizes and summarizes documents hierarchically. Understanding the trade-offs between these methods is key to building robust and scalable RAG systems.

This paper introduces *MyRAG*, an open-source framework designed to compare multiple retrieval and augmented generation strategies systematically. We evaluate embeddings, vector stores, and advanced retrieval enhancements like ColBERT re-ranking and RAPTOR summarization, applying them on the BioASQ [7] and Hugging Face Document QA datasets [8]. Our experiments aim to provide a clearer understanding of how these components influence retrieval accuracy and end-to-end QA performance.

2 Problem Statement

While the Massive Text Embedding Benchmark (MTEB) [9] provides valuable insights into embedding model performance across diverse tasks, there remains a critical need for comprehensive evaluation of end-to-end RAG architectures. The current landscape lacks:

1. **Architecture-Specific Evaluation:** Unlike MTEB's focus on embedding quality, MyRAG provides systematic comparison of different RAG architectures - ColBERT reranking [4], and RAPTOR hierarchical retrieval [5, 6] - using consistent datasets, metrics, vector stores (Chroma and DeepLake) and quantization strategies.

2. **Resource-Conscious Assessment:** MyRAG evaluates both 8-bit quantized and full-precision versions of popular embedding models addressing practical deployment considerations not covered by MTEB's leaderboard.

3. **Intuitive Real-World Metrics:** MyRAG introduces straightforward measures beyond MTEB's technical metrics:

- Correct/Partial/Incorrect answer classification
- Multi-document per question retrieval accuracy
- Response generation quality with context
- Resource utilization across architectures

Through this comprehensive evaluation framework, MyRAG helps practitioners select optimal combinations of embedding models (guided by MTEB's leaderboard), retrieval architectures, and implementation approaches. The framework supports both detailed technical assessment and simple, interpretable metrics that organizations need to optimize their RAG deployments across use cases and resource constraints. By offering comparison of standard RAG, ColBERT reranking, and RAPTOR approaches on multtiple datasets, MyRAG provides actionable insights for building more effective retrieval-augmented generation systems.

MyRAG addresses this need by providing an open-source code framework that enables direct, end-to-end comparisons of RAG approaches. By incorporating multiple embedding models, vector databases, reranking strategy (ColBERT), and hierarchical retrieval architecture (RAPTOR), *MyRAG* provides a clear, cohesive platform for empirical evaluation. This approach ensures that even non-experts can understand performance trade-offs through accessible and practical metrics, ultimately guiding practitioners and researchers toward more optimal and tailored RAG solutions.

3 Methodology

MyRAG builds on the works by offering a unified platform to compare multiple embedding models, vector stores, and rag architectures including ColBERT and RAPTOR and provides a flexible pipeline to switch between embedding models, vector stores, and retrieval techniques. Key components include:

3.1 Parsing Documents and Chunking

MyRAG employs a robust text processing pipeline for document parsing and chunking, leveraging the `RecursiveCharacterTextSplitter` from LangChain [10]. This method enables efficient handling of large textual data by splitting documents into manageable, contextually coherent chunks while preserving overlap for improved information retrieval. The pipeline includes the following key functionalities:

- **Customizable Chunking Parameters:** Users can specify the `chunk_size` (default: 2000 characters) and `chunk_overlap` (default: 250 characters) to control the size and redundancy of chunks. This ensures that relevant information is not lost at chunk boundaries, a critical factor for maintaining context in retrieval-augmented pipelines.
- **Dynamic Splitting Based on Separators:** The chunking process supports a hierarchy of separators, including paragraph breaks (`"\n\n"`), line breaks (`"\n"`), sentence endings (`". "`), and whitespace (`" "`). These separators allow adaptive splitting tailored to the structure of the input text, ensuring logical segmentation while minimizing disruption to the content.
- **Metadata Annotation:** Each chunk is annotated with rich metadata, such as document identifiers, chunk indices, total chunk count, and source type. This metadata facilitates traceability and enables fine-grained analysis of retrieval and generation performance.
- **Integration with PDF Parsing:** The system seamlessly integrates with a custom `PDFLoader` module, which extracts text content from PDF files. This module supports single and batch PDF loading, ensuring compatibility with diverse document formats often encountered in real-world applications.

This sophisticated parsing and chunking approach not only ensures the efficient handling of long-form text but also lays the foundation for effective retrieval and augmentation workflows within the MyRAG framework.

3.2 Embedding Models

MyRAG supports a range of embedding models that are integrated with AWS Bedrock and Hugging Face Transformers [11, 12]. These models translate textual inputs into high-dimensional vector representations, which serve as the foundation for semantic similarity-based retrieval.

- **Hugging Face-Based Models:** Models such as NV-Embed-v2, Stella 1.5B, MXBai Large, and all-MiniLM-L6-v2 are supported via the Hugging Face Transformers library. The implementation includes automatic device management (CPU/CUDA), 8-bit quantization support, and configurable device mapping for efficient resource utilization. Key features include:
  - Automatic normalization of embeddings using L2 normalization
  - Batch processing with customizable batch sizes for memory efficiency
  - Support for instruction-based embedding generation
  - Explicit memory management with CUDA cache clearing and garbage collection
  - Optional model configurations including trust_remote_code and load_in_8bit for quantization and memory optimization
- **Amazon Bedrock Integration:** The Amazon Titan embedding model is accessed through AWS Bedrock's runtime client api.

Each embedding model implementation follows a consistent interface defined by the BaseEmbedding abstract class, ensuring uniform integration with the broader RAG pipeline. The system supports dynamic switching between models and handles cleanup operations to maintain efficient resource usage throughout the embedding process.

These embedding strategies form a core component of MyRAG's retrieval workflow. By evaluating models across benchmark datasets, it becomes possible to determine which embedding configurations yield the best balance of retrieval accuracy, computational efficiency, and downstream QA performance.

3.3 Vector Databases

MyRAG integrates support for Chroma [13] and DeepLake [14], two vector databases specifically designed to optimize similarity search. These databases leverage **Hierarchical Navigable Small World (HNSW)** indices [15], which are highly efficient for approximate nearest neighbor search. HNSW organizes vectors in a spatial domain into hierarchical clusters, enabling scalable similarity comparisons.

The key functionality of HNSW is its ability to cluster vectors in n-dimensional space based on proximity. For instance, given a dataset of 5000 vectors, HNSW groups similar vectors into clusters, each represented by a centroid or a representative vector. Assuming an average cluster size of 100 vectors, this process reduces the search space to 50 representative vectors. Consequently, the computational complexity of similarity comparisons is significantly reduced, shifting from 5000 pairwise comparisons to just 50. This hierarchical organization ensures efficient retrieval without compromising accuracy.

In addition to HNSW indices, both Chroma and DeepLake support similarity metrics like cosine and L_2 distance, making them versatile for a wide range of use cases. Furthermore, these vector databases handle rich metadata alongside the vector representations, allowing for more

nuanced queries and contextual filtering. These features collectively enhance the performance and scalability of retrieval pipelines within the MyRAG framework.

3.4 LLMs

MyRAG supports integration with both AWS Bedrock and Hugging Face Transformers-based models for text generation [11, 12]. These Large Language Models (LLMs) serve as the backbone of the retrieval-augmented generation workflow, consuming retrieved documents as context and producing coherent, contextually grounded responses. The system implements a common interface through the BaseLLM abstract class, enabling seamless switching between different LLM backends.

- **AWS Bedrock LLMs:** Models such as Anthropic's Claude variants are accessed through AWS Bedrock's runtime client API.
- **Hugging Face Transformers LLMs:** Integration of models like Meta-Llama-3-8B-Instruct with features including:
  - **Resource Management:**
    - Automatic CPU/CUDA device selection
    - 8-bit quantization support
    - Configurable device mapping for distributed inference
    - Automatic CUDA cache clearing and garbage collection

The system maintains consistent cleanup procedures across implementations, ensuring efficient resource management particularly important for GPU-based deployments. This standardized approach enables systematic evaluation of different LLM configurations within the broader RAG pipeline.

3.5 ColBERT Reranking

ColBERT [4] provides a reranking framework that refines retrieval results by comparing query tokens against all tokens in candidate documents, rather than relying on a single vector embedding per document. Unlike traditional embeddings, which compress each document into a single vector, ColBERT applies a late interaction mechanism to handle token-level embeddings. This approach computes similarities between each query token embedding and every token embedding in a document, ensuring that important context within a document is not lost due to over-compression.

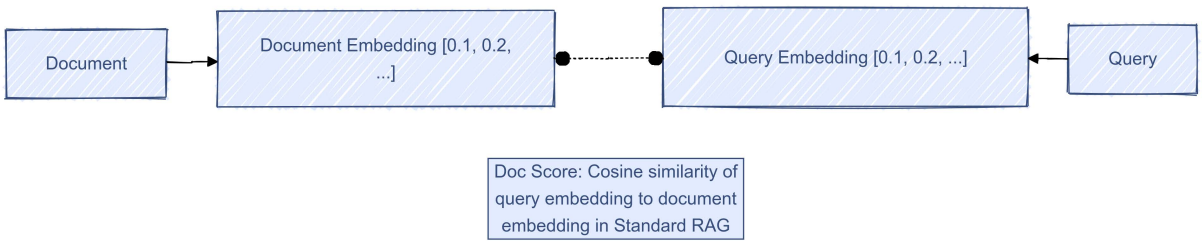

Figure 2: Flow chart of Standard RAG Embeddings Document Scoring within vector database

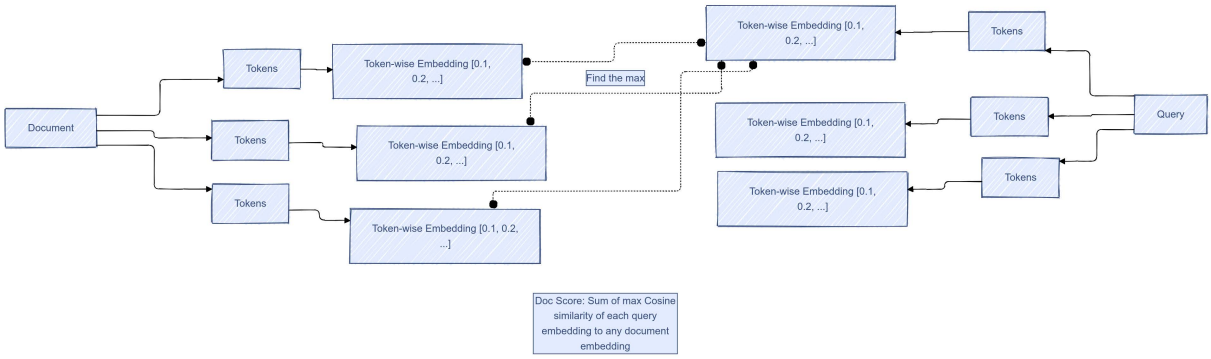

Figure 3: Flow chart of ColBert Document Scoring

The implementation in MyRAG leverages the RAGatouille library [16] to integrate ColBERT v2.0 as a re-ranking step following the initial retrieval process. After retrieving a candidate set of documents, the RAGatouille-based ColBERT implementation scores each document based on fine-grained token alignments with the query. These fine-grained comparisons enable the system to re-prioritize documents that contain subtle but crucial semantic clues. The refined ranking supports more accurate downstream retrieval-augmented generation (RAG) tasks, enhancing both precision and robustness. Through this method, MyRAG gains the ability to emphasize relevant sections of documents that might otherwise be overshadowed by broad embedding representations.

The re-ranking step is implemented as a modular pipeline component, allowing for easy integration with different retrieval configurations. This modularity enables MyRAG to compare performance with and without reranking, providing insights into the impact of fine-grained token matching on overall system performance. Figures below shows how ColBert embedding document scoring for top K differs from standard rag and how it is integrated in the MyRAG pipeline

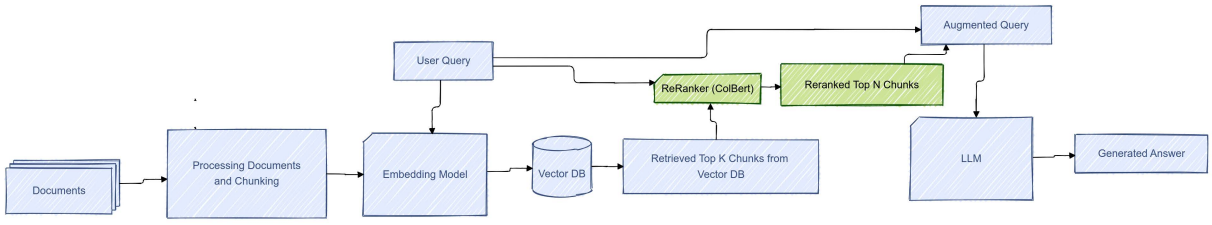

Figure 4: Flow chart of Standard Rag with ColBert Reranking as integrated in MyRag

3.6 RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval

RAPTOR [5, 6] implements a hierarchical retrieval architecture that organizes documents into a multi-level tree structure. Our implementation adapts the open-source code from the RAPTOR 2024 paper [6], particularly their approach to global and local clustering hierarchies. The implementation consists of several key components:

1) **Embedding and Dimensionality Reduction:** Following the RAPTOR architecture, documents are first embedded using a chosen embedding model. The system then employs UMAP (Uniform Manifold Approximation and Projection) for both global and local dimensionality reduction, with different neighborhood parameters.

2) **Cluster Formation:** Adopting the RAPTOR paper's clustering strategy, the system uses Gaussian Mixture Models (GMM) with Bayesian Information Criterion (BIC) to automatically determine the optimal number of clusters. Documents can belong to multiple clusters based on a probability threshold, allowing for overlapping topic coverage. The number of clusters is dynamically optimized up to a maximum of 50, adapting to the document structure.

3) **Hierarchical Summarization:** At each level, documents within the same cluster are summarized using a Language Model (LLM). The summarization process is recursive, with each level building upon the summaries from the previous level. This creates a hierarchy of increasingly abstract representations, with the number of levels configurable through the `n_levels` parameter.

4) **Integration with RAG:** The system maintains both the original documents and their summaries in the vector store, enabling flexible retrieval at different levels of abstraction. This allows the system to match queries with either specific details from source documents or broader thematic summaries, depending on the query's nature.

The implementation employs RAPTOR's clustering strategies, with careful memory management and batch processing to handle large document collections efficiently. This approach enables the system to capture both fine-grained relationships between similar documents and broader thematic connections across the corpus.

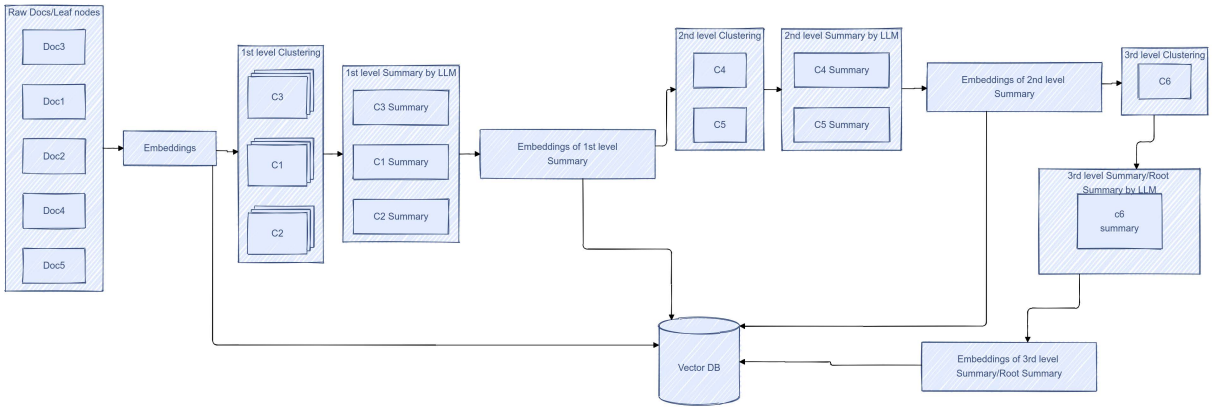

Figure 5: Flow chart of RAPTOR processing of documents as integrated in MyRAG

3.7 Pipeline Implementation

The retrieval-augmented generation pipeline is implemented as a modular sequence of clearly defined steps, each encapsulated in a `PipelineStep` class. This design allows components such as document processing, embedding generation, retrieval, reranking, and LLM-based generation that were discussed above to be easily composed, replaced, or extended.

1. **Data Abstraction:** All information (documents, embeddings, queries, metadata, and results) is passed through the pipeline in a centralized `PipelineData` object. This shared state container prevents the need for extensive inter-step dependencies.

2. **Document Processing and Embedding:** The pipeline begins by applying the previously described chunking and metadata annotation to raw documents via a `DocumentProcessor`. Next, a `DocumentEmbedder` step uses the selected embedding model to encode these chunks into vector representations.

3. **Query Embedding:** A `QueryEmbedder` step similarly encodes input queries into vector form, ensuring consistency between document and query representations.

4. **Retrieval and Reranking:** Using the chosen vector database and similarity search as described above, the `Retriever` step identifies the top-k relevant chunks. If ColBERT-based

reranking is enabled, a subsequent `RerankerStep` refines the ranking of these retrieved chunks, emphasizing token-level alignments for improved relevance.

5. **LLM Generation:** Finally, the `Generator` step forwards both the retrieved (and optionally reranked) contexts and the original query to the LLM for response generation, resulting in a final answer that leverages the combined knowledge from the external documents.

6. **RAPTOR Option:** When using RAPTOR, the pipeline replaces the standard `DocumentProcessor` with a `RaptorProcessor`, which recursively clusters and summarizes content at multiple hierarchy levels before embeddings and retrieval. This ensures scalable handling of large corpora and improved retrieval focus.

This modular composition, defined as a `RAGPipeline`, enables experimentation and fine-grained comparison across various embedding models, LLMs, retrieval strategies, and architectures like RAPTOR. The code's design ensures easy configuration, reproducibility, and extension, supporting a wide array of use cases in retrieval-augmented generation.

3.8 Demo GUI

MyRAG includes an interactive web interface built with Streamlit that allows users to experiment with different RAG Pipeline configurations. The GUI offers two main interfaces:

- **Simple RAG:** Allows testing of standard RAG architecture with configurable parameters including:
  - Choice of embedding models (NV-Embed-v2, Stella 1.5B, MXBai Large, etc.)
  - LLM selection (Claude 3.5 Sonnet, Llama 3 8B Instruct)
  - Customizable system messages
  - Adjustable pipeline parameters (chunk size, overlap, retrieved documents k, etc.)
- **RAG with Reranking:** Extends the simple RAG interface with ColBERT reranking capabilities, maintaining all base configuration options while adding reranking-specific parameters.

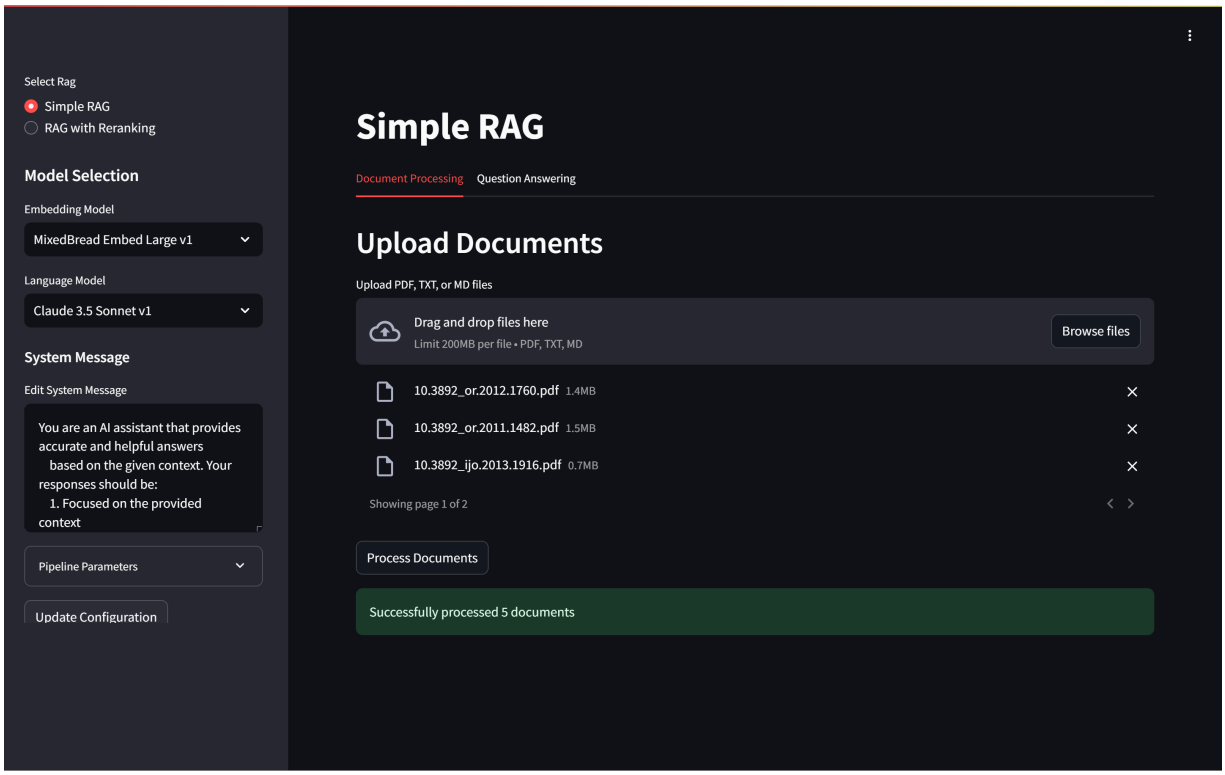

Figure 6: MyRAG Process Documents with Simple RAG

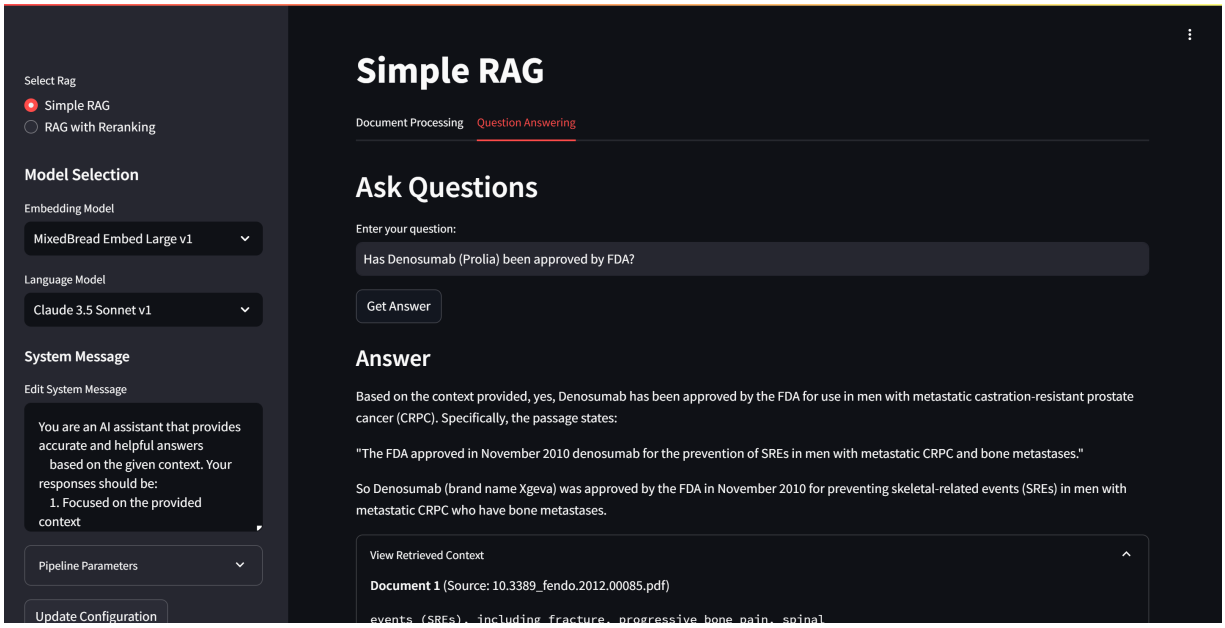

Figure 7: MyRAG Question-Answering with Simple RAG

RAPTOR implementation is in progress and will be updated soon. The interface provides visualization of the MyRAG pipeline's performance.

4 Evaluations and Results

4.1 Datasets

1. Hugging Face Document QA Evaluation Dataset(HF QA):

The *huggingface_doc_qa_eval* dataset is a synthetic dataset consisting of question-answer pairs extracted from the markdown files of the Hugging Face repository on GitHub. It is specifically designed for evaluating Retrieval-Augmented Generation (RAG) systems [8]. The dataset contains a total of 65 unique questions, each paired with its corresponding answer, context, and source document path.

Out of the ten available columns in the dataset, the following four were used for evaluation:

- **Context:** The content retrieved from the corresponding markdown file.
- **Question:** The query posed to the system.
- **Answer:** The ideal answer to the question.
- **Source_doc:** The path to the markdown file in the Hugging Face GitHub repository.

The dataset was loaded in Parquet format using the Hugging Face API. Each question-answer pair was derived from a unique markdown file, ensuring there was no overlap in the source documents. This design allows for precise evaluation of RAG systems in retrieving relevant content from specific documents.

2. BioASQ11 Challenge Dataset(BioASQ):

The *BioASQ11* dataset was obtained from the 2023 iteration of the BioASQ challenge, specifically training set (11b) of Task Set B for biomedical question answering [7]. The training set (11b) consists of a total of 4,719 questions, each provided with gold-standard annotations, including concepts, article URLs, snippets, RDF triples, "exact" answers, and "ideal" answers.

Due to the size of the dataset and restrictions on accessing articles from certain URLs, a filtering process was applied:

- Articles that required credentials for access were excluded.
- To comply with server limitations on large-scale downloads, only a subset of 13 questions with 42 associated PDFs was retained for the evaluation.

For each retained question, answers are derived from 1 to 7 different PDFs. To facilitate evaluation, a CSV file was created for this filtered questions with the following columns:

- **Question ID:** A unique identifier for each question.
- **Question:** The query posed.
- **Ideal Answer:** The gold-standard answer for the question.
- **Download Link:** The URL for downloading the article PDF.
- **PDF Reference:** The specific reference to the PDFs used for the answer.

A web scraping script was developed to retrieve PDF file URLs from the provided article links. This processed dataset enables systematic evaluation of retrieval accuracy and relevance in biomedical contexts.

4.2 TOP-K Retrieval Accuracies

Retrieval accuracy for each embedding model is evaluated using a standard top-k retrieval metric. Given N queries, let retrieved$_i$@k denote the set of top-k retrieved chunks document ids for the i-th query, and let relevant$_i$ represent the set of relevant document ids for that query. The top-k retrieval accuracy, Accuracy@k, is computed as follows:

$$\text{Accuracy@k} = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}\mathbb{1}(\exists d \in \text{retrieved}_i@k : d \in \text{relevant}_i), & \text{if } |\text{relevant}_i| = 1 \text{ or } k = 1\\ \frac{|\text{retrieved}_i@k \cap \text{relevant}_i|}{\min(k, |\text{relevant}_i|)}, & \text{otherwise}\end{cases} \tag{1}$$

The evaluations are conducted on both the BioASQ and Hugging Face Document QA datasets(HF QA). All runs use the common retrieval parameters shown in Table 1. Each query is processed using a given embedding model, and the resulting top-k candidates are examined according to the equation above.

Retrieval Parameter	**Value**
Chunk Size	4000 tokens
Chunk Overlap	200 tokens
Top-k	1,2,3,4,5,7,8,10
Query Instruction for query embedding generation	**NV-Embed-v2**: "Instruct: Given a question, retrieve passages that answer the question. Query:" **stella_en_1.5B_v5** : "Instruct: Given a web search query, retrieve relevant passages that answer the query. Query:" no instructions were used for queries embedding with other embedding models

Table 1: Common retrieval parameters and instructions used across experiments.

Table 2 presents the model specifications. The `mxbai-embed-large-v1` model is extremely resource-efficient and still provides competitive performance, making it suitable for environments with limited computational capacity. In contrast, `NV-Embed-v2 (8-bit)` is notably larger, yet still manageable due to quantization, and it achieves performance comparable to much larger models. The `stella_en_1.5B_v5` model is larger and performs slightly better than `NV-Embed-v2 (8-bit)`, but at the cost of higher resource usage. Despite its closed-source nature and unknown parameters, `titan-embed-text-v2` does not surpass the best open-source embeddings. The `all-MiniLM-L6-v2` model, while very lightweight, trails behind the top performers.

Embedding Model	**Quantization**	**Size (GB)**	**Parameters (M)**
mxbai-embed-large-v1	None	1.25	335.14
stella_en_1.5B_v5 (8-bit)	8-bit	3.48	1543.27
stella_en_1.5B_v5	None	9.25	1543.27
NV-Embed-v2 (8-bit)	8-bit	7.44	7851.02
all-MiniLM-L6-v2	None	0.08	22.71
titan-embed-text-v2	Unknown	Unknown	Unknown

Table 2: Specifications of embedding models evaluated.

Table 3 shows the top-k retrieval accuracy under standard RAG (no reranking). It can be observed that `NV-Embed-v2 (8-bit)`, `mxbai-embed-large-v1`, and `stella_en_1.5B_v5` achieve strong accuracy scores. Although `NV-Embed-v2 (8-bit)` is large, quantization allows it to be

memory-efficient. The `mxbai-embed-large-v1` model stands out as both lightweight and high-performing, while `all-MiniLM-L6-v2` remains light weight but lags in accuracy. The closed-source `titan-embed-text-v2` does not outperform the open-source models.

Model	Dataset	k=1	k=2	k=3	k=4	k=5	k=7	k=8	k=10
NV-Embed-v2 (8-bit)	BioASQ	1.00	0.85	0.79	0.76	0.77	0.79	0.80	0.80
	HF QA	0.91	0.94	0.98	0.98	0.98	1.00	1.00	1.00
all-MiniLM-L6-v2	BioASQ	0.62	0.58	0.51	0.47	0.57	0.54	0.57	0.61
	HF QA	0.58	0.66	0.72	0.74	0.77	0.80	0.82	0.83
mxbai-embed-large-v1	BioASQ	1.00	0.92	0.77	0.70	0.74	0.77	0.79	0.79
	HF QA	0.95	0.95	0.95	0.95	0.95	0.98	0.98	0.98
stella_en_1.5B_v5	BioASQ	1.00	0.73	0.72	0.71	0.72	0.74	0.75	0.79
	HF QA	0.95	0.98	0.98	0.98	1.00	1.00	1.00	1.00
stella_en_1.5B_v5 (8-bit)	BioASQ	0.31	0.42	0.64	0.66	0.68	0.74	0.74	0.78
	HF QA	0.88	0.91	0.91	0.91	0.92	0.94	0.94	0.94
titan-embed-text-v2	BioASQ	1.00	0.81	0.69	0.67	0.69	0.67	0.68	0.73
	HF QA	0.89	0.91	0.95	0.97	0.98	0.98	0.98	0.98

Table 3: Top-k retrieval accuracy for standard RAG. All evaluations are performed with the parameters and instructions listed in Table 1.

ColBERT v2.0 reranking is then introduced to refine the top-K results from an initial retrieval of 15 chunks. The reranking parameters used are shown in Table 4, and Table 5 presents the corresponding accuracy scores. The ColBERT reranking model, with a size of only 0.41 GB and 110M parameters, quickly rescored candidates without significant overhead.

Reranking Parameter	Value
Initial Retrieval Top k	15
Reranking Top k	1,2,3,4,5,7,8,10
Chunk Size	4000 tokens
Chunk Overlap	200 tokens
Query Instruction for query embedding generation	**NV-Embed-v2**: "Instruct: Given a question, retrieve passages that answer the question. Query:" **stella_en_1.5B_v5** : "Instruct: Given a web search query, retrieve relevant passages that answer the query. Query:" no instructions were used for queries embedding with all other embedding models including reranking ColBERT v2.0 embedding model

Table 4: Parameters used for experiments involving ColBERT v2.0 Reranking.

Model	Dataset	k=1	k=2	k=3	k=4	k=5	k=7	k=8	k=10
NV-Embed-v2 (8-bit)	BioASQ	1.00	0.81	0.74	0.77	0.78	0.80	0.80	0.82
	HF QA	0.97	0.98	0.98	0.98	0.98	0.98	1.00	1.00
all-MiniLM-L6-v2	BioASQ	0.85	0.77	0.72	0.67	0.69	0.72	0.72	0.73
	HF QA	0.88	0.89	0.89	0.89	0.91	0.91	0.91	0.91
mxbai-embed-large-v1	BioASQ	1.00	0.88	0.77	0.73	0.74	0.80	0.82	0.82
	HF QA	0.92	0.97	0.97	0.98	0.98	0.98	0.98	0.98
stella_en_1.5B_v5	BioASQ	1.00	0.85	0.72	0.67	0.73	0.79	0.79	0.79
	HF QA	0.95	0.98	0.98	0.98	0.98	0.98	0.98	1.00
stella_en_1.5B_v5 (8-bit)	BioASQ	0.92	0.81	0.72	0.67	0.68	0.69	0.70	0.72
	HF QA	0.77	0.77	0.77	0.77	0.78	0.78	0.78	0.82
titan-embed-text-v2	BioASQ	1.00	0.85	0.74	0.71	0.73	0.75	0.77	0.79
	HF QA	0.92	0.95	0.95	0.95	0.95	0.95	0.97	0.98

Table 5: Top-k retrieval accuracy after ColBERT reranking.

ColBERT reranking generally improves performance for models that had weaker initial retrieval results. For instance, `all-MiniLM-L6-v2` and `titan-embed-text-v2` show notable gains, indicating that fine-grained token-level scoring helps surface more relevant candidates at top ranks. For strong models like `NV-Embed-v2 (8-bit)`, reranking raises the HF QA $k = 1$ accuracy from 0.91 to 0.97, but the gains diminish at higher k values, suggesting that reranking is most beneficial for improving the very light weight models.

The RAPTOR-based retriever is not included in these top-k evaluations as it involves hierarchical summaries from multiple documents that complicate direct comparison to standard retrieval methods. Future iterations will incorporate RAPTOR retrieval accuracies, enabling comparisons across the full spectrum of retrieval strategies.

4.3 RAG Evaluation

The end-to-end effectiveness of the Retrieval-Augmented Generation (RAG) pipeline is assessed by integrating the retrieved document chunks into a Large Language Model (LLM) for final answer generation. All experiments described here use Anthropic's Claude 3.5 Sonnet (20240620-v1) model accessed through AWS Bedrock. Preliminary tests with Meta-Llama-3-8B-Instruct (8-bit) indicated insufficient answer quality and limited computational resources, so the evaluations focus exclusively on the Claude.

Table 6 presents the performance of standard RAG across each embedding model. The system message and generation configuration remain consistent in all experiments, as defined in the evaluation configuration files. This generation configuration ensures that the same input produces the same deterministic response under identical conditions. The parameters are:

- System Message:

```
You are an AI assistant that provides accurate and helpful answers
based on the given context. Your responses should be:
1. Focused on the provided context
2. Clear and concise
3. Accurate and relevant to the question
4. Based only on the information given
```

- Generation Config:

```
max_tokens: 512
temperature: 0.1
top_k: 50
top_p: 0.9
anthropic_version: bedrock-2023-05-31
```

The standard RAG evaluation uses $TOP\ k\ chunk\ retrieved = 5$, $chunk\_size = 4000$, $chunk\_overlap = 200$ and Query Instructions same as done in top k retrieval.

Embedding Model	Dataset	Total	Correct	Incorrect	Partial	Correct (%)
NV-Embed-v2 (8-bit)	HF QA	65	64	1	0	98.46
	BioASQ	13	9	1	3	69.23
all-MiniLM-L6-v2	HF QA	65	50	15	0	76.92
	BioASQ	13	6	3	4	46.15
mxbai-embed-large-v1	HF QA	65	62	3	0	95.38
	BioASQ	13	8	1	4	61.54
stella_en_1.5B_v5	HF QA	65	64	1	0	98.46
	BioASQ	13	9	0	4	69.23
titan-embed-text-v2	HF QA	65	63	2	0	96.92
	BioASQ	13	8	0	5	61.54

Table 6: End-to-end RAG performance under standard retrieval conditions.

Under standard RAG, both NV-Embed-v2 (8-bit) and Stella EN 1.5B v5 achieve high correctness percentages in HF QA, exceeding 98% correct responses. The mxbai-embed-large-v1 and titan-embed-text-v2 models also perform well, surpassing 95% correctness on HF QA. The all-MiniLM-L6-v2 model yields lower correctness, especially on BioASQ. For more specialized domains like BioASQ, performance generally declines compared to HF QA (each answer from a single document), reflecting the increased difficulty of biomedical questions from multiple documents.

Next, Table 7 presents the results of standard RAG augmented with ColBERT reranking. The reranking configuration uses $max\_k = 15$ for initial retrieval and $rereank\_max\_k = 5$ for the final selection. All other parameters, including the generation configuration, remain consistent. ColBERT reranking did not significantly change correctness percentages for most models. While token-level reranking refines top document selection, it does not necessarily translate into substantially improved final answer correctness. The all-MiniLM-L6-v2 model shows some gains on HF QA, improving from 76.92% to 89.23%, but on BioASQ performance declines further, highlighting the complexity of biomedical queries.

Embedding Model	Dataset	Total	Correct	Incorrect	Partial	Correct (%)
NV-Embed-v2 (8-bit)	HF QA	65	63	2	0	96.92
	BioASQ	13	8	0	5	61.54
all-MiniLM-L6-v2	HF QA	65	58	7	0	89.23
	BioASQ	13	5	3	5	38.46
mxbai-embed-large-v1	HF QA	65	63	2	0	96.92
	BioASQ	13	8	0	5	61.54
stella_en_1.5B_v5	HF QA	65	63	2	0	96.92
	BioASQ	13	8	0	5	61.54
titan-embed-text-v2	HF QA	65	62	3	0	95.38
	BioASQ	13	8	0	5	61.54

Table 7: End-to-end RAG performance with ColBERT reranking.

Finally, Table 8 reports the results when using the RAPTOR hierarchical summarization-based retrieval architecture. RAPTOR uses a summarization template designed to handle multiple documents effectively:

```
Here is a sub-set of documents.
Give a detailed summary of the content provided.

Documents:
```

Embedding Model	Dataset	Total	Correct	Incorrect	Partial	Correct (%)
mxbai-embed-large-v1	HF QA	65	64	1	0	98.46
	BioASQ	13	9	1	3	69.23
stella_en_1.5B_v5	HF QA	65	65	0	0	100.00
	BioASQ	13	10	0	3	76.92

Table 8: End-to-end RAG performance using the RAPTOR architecture.

RAPTOR excels at managing complex, multi-document queries. As shown in Table 8, Stella EN 1.5B v5 with RAPTOR achieves perfect correctness (100%) on HF QA and 76.92% correctness on BioASQ, outperforming both the standard and reranked RAG configurations. The mxbai-embed-large-v1 model also shows improvement under RAPTOR. Due to computational resource limitations and maximum token constraints, other embedding models, could not be tested in this RAPTOR architecture. Nevertheless, the RAPTOR-based approach demonstrates promising gains in challenging domains like biomedical QA, where detailed reasoning and synthesis from multiple sources are crucial.

5 Challenges and Future Work

- **Evaluation Metrics:** Current retriever evaluation focuses on document-level accuracy. Future work includes more fine-grained evaluation (e.g., chunk level), automated scoring methods for retriever, and domain expert assessments of entire QA RAG to better understand retrieval quality in specialized domains.
- **Document Parsing:** MyRAG currently processes textual data. Extending it to multimodal inputs (PDFs with figures, images, or even audio transcripts) would demand integrating vision and ASR models, enabling comprehensive retrieval across diverse data formats.
- **Multi-agent Systems:** A future direction involves multi-agent LLM architectures where one agent refines user queries and another specializes in retrieval. Iterative query clarification and refinement would ensure that only the most relevant information is retrieved for the LLM to answer.

6 Conclusion

This work introduced MyRAG, an open-source code for systematically comparing a variety of Retrieval-Augmented Generation (RAG) architectures, embedding models, vector databases, and retrieval enhancement techniques. The experiments demonstrated that even lightweight embedding models, such as mxbai-embed-large-v1, can perform competitively in top-k retrieval accuracy, offering efficient solutions suitable for resource-constrained environments. While larger models like NV-Embed-v2 (8-bit) and Stella EN 1.5B v5 provided slightly better retrieval accuracy, the differences were marginal. ColBERT reranking generally improved retrieval accuracy for models with weaker initial performance, though its benefits diminished at higher values of k. This suggests that while token-level reranking can refine top candidates, it may not guarantee significant gains across broader retrieval sets.

The end-to-end RAG evaluation provided further insight into how embedding models and retrieval methods influence final answer quality. Standard RAG delivered strong correctness on simpler tasks, while more complex and domain-specific questions, such as those from BioASQ, proved more challenging. ColBERT reranking led to moderate improvements in certain cases (e.g., all-MiniLM-L6-v2 on HF QA), but did not universally enhance final answer accuracy. In contrast, the RAPTOR hierarchical retrieval architecture offered substantial improvements for difficult, multi-document BioASQ queries.

These findings highlight the importance of selecting the right combination of embedding model, retrieval strategy, and architecture to achieve optimal RAG performance. Future work will focus on more granular retrieval evaluations, integrating multimodal data, and exploring multi-agent LLM architectures. By continually refining RAG pipelines and incorporating new advancements, MyRAG aims to serve as a versatile and evolving platform for building more effective and efficient retrieval-augmented systems.

Acknowledgments: The author extends sincere gratitude to Abdygulov Timur for his guidance in understanding and developing the standard RAG, ColBERT, and RAPTOR architectures, as well as for assisting in preparing the BioASQ dataset for evaluation. The author also thanks Professor Amir H. Jafari for helping understand RAG concept, sharing literature related to RAG, introducing the BioASQ dataset, and for his impactful NLP and Deep Learning courses, which greatly contributed to the successful implementation of the methods described in this work.

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021. [Online]. Available: https://arxiv.org/abs/2005.11401

[2] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, "Realm: Retrieval-augmented language model pre-training," 2020. [Online]. Available: https://arxiv.org/abs/2002.08909

[3] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *arXiv preprint arXiv:2307.03172*, 2023.

[4] O. Khattab and M. Zaharia, "Colbert: Efficient and effective passage search via contextualized late interaction over bert," 2020. [Online]. Available: https://arxiv.org/abs/2004.12832

[5] J. Wu, L. Ouyang, D. M. Ziegler, N. Stiennon, R. Lowe, J. Leike, and P. Christiano, "Recursively summarizing books with human feedback," 2021. [Online]. Available: https://arxiv.org/abs/2109.10862

[6] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, "Raptor: Recursive abstractive processing for tree-organized retrieval," 2024. [Online]. Available: https://arxiv.org/abs/2401.18059

[7] A. Krithara, A. Nentidis, K. Bougiatiotis, and G. Paliouras, "Bioasq-qa: A manually curated corpus for biomedical question answering," 2022. [Online]. Available: https://doi.org/10.1101/2022.12.14.520213

[8] "Huggingface doc qa eval dataset," https://huggingface.co/datasets/m-ric/huggingface_doc_qa_eval.

[9] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "Mteb: Massive text embedding benchmark," *arXiv preprint arXiv:2210.07316*, 2022. [Online]. Available: https://arxiv.org/abs/2210.07316

[10] LangChain, "Langchain documentation," 2024. [Online]. Available: https://python.langchain.com/

[11] "HuggingFace Docs," https://huggingface.co/docs.

[12] *Amazon Bedrock API Reference*, Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/bedrock/latest/APIReference/welcome.html

[13] Chroma, "Chroma," https://docs.trychroma.com/, 2023.

[14] deeplake, "deeplake," https://docs.deeplake.ai/en/latest/deeplake.html, 2022.

[15] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *arXiv preprint arXiv:1603.09320*, 2016. [Online]. Available: https://arxiv.org/abs/1603.09320

[16] B. Clavié, "Ragatouille," GitHub, 2024. [Online]. Available: https://github.com/AnswerDotAI/RAGatouille