International Journal of Advance and Applied Research
www.ijaar.co.in
ISSN – 2347-7075 Impact Factor – 8.141
Peer Reviewed Bi-Monthly
Vol. 6 No. 22 March - April - 2025

RAG In Research: Exploring The Intersection of Retrieval and Generation

Vanita B. Mhaske[1] & Rhishikesh S. Kadam[2]
[1]*Department of Computer Science, PVGCOSC, Pune, India.*
[2]*Independent Researcher Pune, India.*
Corresponding Author – Vanita B. Mhaske
DOI - 10.5281/zenodo.15533920

Abstract:

In today's world of information overload, it's essential to manage data efficiently and get useful insights from various sources. Retrieval-Augmented Generation (RAG) systems assist by combining methods to retrieve relevant information with models that generate clear and precise answers based on that information. This approach makes it easier to access and understand the information we need. By breaking documents into chunks and embedding them, the system enables fast and targeted retrieval of relevant information from large document collections, saving time compared to reading full documents. Chunks preserve contextual boundaries within the text, so retrieved information retains coherence and relevance, helping the generative model (like LLaMA 3) create meaningful answers without losing essential context. RAG enhances the generative power of language models by integrating information retrieval, enabling more accurate and contextually rich responses in natural language query systems. We have access to a vast amount of research data, which greatly contributes to our knowledge. However, it is not always feasible to read every paper to answer specific questions. To address this, we implemented Retrieval-Augmented Generation (RAG), a system that allows you to load relevant research papers and ask any questions related to the topics covered in those documents. This enables efficient access to information without needing to manually review each paper. To build this system, we use the Sentence Transformer and LLaMA 3 model.

Keywords: Retrieval-Augmented Generation (RAG), Question-Answering System, Information Retrieval, Language Generation, Similarity Search, Embeddings, Vector Database

Introduction:

RAG is basically a retrieval system through which we can retrieve content from a single or multi document. There are already many more techniques which are helpful. These include Traditional Information Retrieval (IR), Embedding- based, Retrieval (Dense Retrieval), Sequence-to-Sequence Models, Transformers (e.g., BERT, GPT, T5, BART), Question Answering (QA) Systems, Summarization Models, Topic Modeling, Text Classification, Named Entity Recognition (NER). but there are some drawbacks too. Many are Struggling with semantic understanding [1] , some require more computational power than traditional IR techniques [6]. Some require fine-tuning, and can sometimes generate irrelevant or incorrect information [7]. QA may fail to answer questions that require deeper reasoning [11]. Extractive methods may not provide accurate answer, and abstract methods can sometimes create incorrect or incomplete summaries [4]. Each technique has its own strengths, weaknesses, and best-use scenarios. Techniques like RAG, try to overcome the problem. Multi-Document Question Answering (MDQA) is an NLP task that involves answering questions based on a collection of multiple documents on the same topic, using a language model to

process and synthesize relevant information from the documents to generate accurate and coherent answers.

We are surrounded by an ever-expanding body of research data, which continuously adds to our collective knowledge. However, with the sheer volume of information available, it becomes increasingly challenging to read every single paper to find answers to specific questions. Manually sifting through hundreds or thousands of papers for relevant information is time-consuming and inefficient, especially when the information is dispersed across multiple sources.

To address this challenge, we implemented RAG– an advanced system designed to streamline the process of accessing research insights. RAG combines information retrieval techniques with generative models, allowing you to load a set of relevant research papers into the system. Once these documents are loaded, users can ask any question related to the topics covered in the papers. The system uses the knowledge from the papers to retrieve relevant content and generate accurate, context-aware responses.

This approach eliminates the need to read entire documents, as the system efficiently identifies and retrieves only the information that is most pertinent to the query. With RAG, users can gain quick and precise answers, making research more accessible and less time-consuming. It enables researchers, students, and professionals to interact with vast amounts of information in a more intuitive and efficient way, leveraging the full potential of modern language models and retrieval techniques.

1. RAG System:

The RAG system is an advanced framework that combines information retrieval and text generation to enhance the ability of language models to handle knowledge-intensive tasks. The process can be broken down into a pipeline that integrates document ingestion, embedding creation, embedding storage in a vector database, querying through information retrieval techniques, and answer generation using a language model.

Document Ingestion:

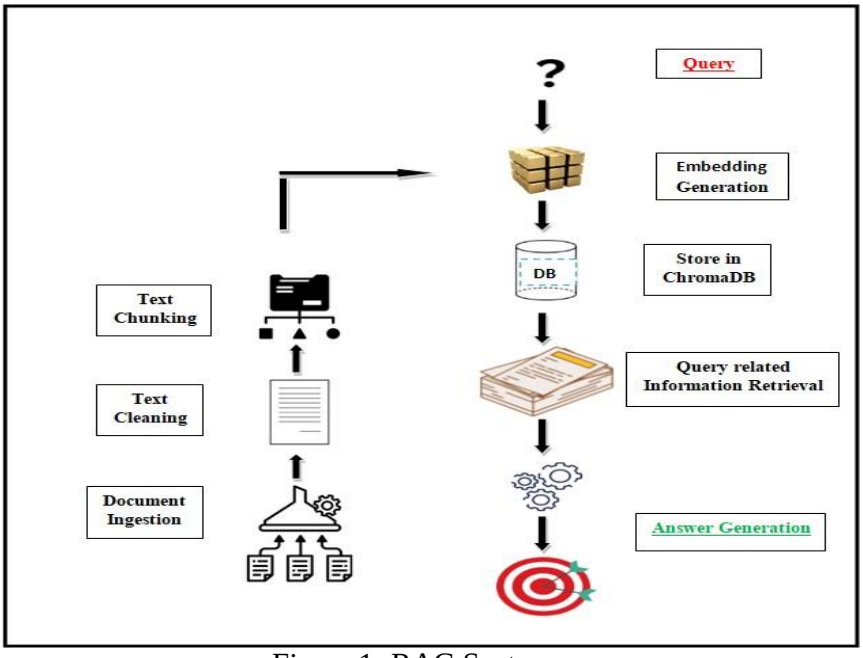

Figure 1: RAG System

The first step involves loading all relevant documents into the system. These documents can be in various formats such as text files, PDFs, or other types of documents. The content of these documents is extracted to make it available for further processing in the system.

Cleaning and Preprocessing:

After ingestion, the text is cleaned to remove unwanted elements, such as special characters, headers, footers etc. Preprocessing may also involve standardizing the text by performing operations like tokenization, which breaks the text into smaller components (e.g., words or sentences tokenization), and normalization, which ensures consistency across the data by, for example, converting text to lowercase.

Text Chunking:

In this step, the cleaned and preprocessed text is divided into smaller chunks or segments. These chunks can be created based on specific size limits or natural boundaries within the text (e.g., paragraphs or sentences). The key goal is to break the content into smaller, manageable pieces that maintain the context of the original text, ensuring that information stays relevant within each chunk.

Embedding Generation:

Each text chunk is then converted into a numerical vector representation, known as an embedding. This transformation captures the semantic meaning of the text, allowing it to be represented as a point in a high-dimensional space. The embedding process enables the system to handle text in a way that supports efficient similarity searches during queryprocessing.

Storing Embeddings:

The generated embeddings are stored in a vector database along with metadata that provides additional context about the source of the content, such as the document from which the chunk was taken. This enables the system to quickly access and retrieve relevant embeddings during the search process, making the retrieval step faster and more efficient.

Information Retrieval:

When a user submits a query, the system generates an embedding for the query and compares it to the embeddings stored in the vector database. Using similarity measures, such as cosine similarity, the system identifies the most relevant text chunks that are similar to the query. The top relevant chunks are retrieved to ensure the response is based on the most pertinent information available.

Answer Generation:

After retrieving the relevant chunks, the system passes this information to a generative language model. The model synthesizes the content from the retrieved chunks and generates a coherent, contextually relevant answer to the user's query. This response is crafted to be informative and aligned with the context of the retrieved data, ensuring that the answer is accurate and useful.

2. Procedure:

Step 1: Document Ingestion:

In this step, we ingest multiple research paper PDFs by first loading each document from the specified directory. These documents are parsed and stored in a structured format as Document objects, each containing the raw page content and essential metadata (e.g., document source, title, publication year). The document ingestion process ensures that all relevant research papers are captured and prepared for further processing, allowing the system to handle diverse types of content efficiently. The documents are then made accessible for subsequent steps, such as cleaning, chunking, and embedding creation. Below are the five research papers used for this process.

Table 1: Research Papers on Deep Learning Models

Sr. No.	Research Paper Name
1	A Comprehensive Overview and Comparative Analysis on Deep Learning Models [8]
2	Attention Is All You Need [10]
3	Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks [2]
4	Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications [5]
5	Sequence to Sequence Learning with Neural Networks [9]

Step 2: Text Cleaning and pre-processing:

In this step, the raw content of the loaded research papers is cleaned to ensure high-quality text for further processing. The cleaning process involves removing unnecessary whitespace, special characters, non-alphanumeric symbols, and other irrelevant elements from the text. Additionally, redundant spaces and formatting issues are addressed to ensure the text is properly structured and consistent. This step improves the quality of the text, making it suitable for chunking, embedding creation, and other subsequent tasks.

Step 3: Documents Chunking:

After cleaning, the documents are split into smaller, manageable chunks of text to facilitate efficient processing. A method like Recursive Text Splitter is used to segment each document into chunks, ensuring each chunk maintains a clear context. To avoid losing meaning at chunk boundaries, an overlap of 50 tokens is added between adjacent chunks, preserving continuity across sections. Metadata, such as the document's filename, title, or publication details, is attached to each chunk, allowing easy tracking of its source. This chunking process helps the system handle large texts effectively while retaining key contextual links.

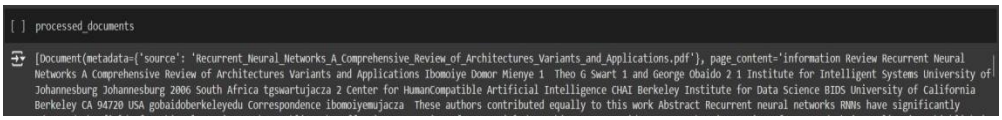

Figure 2: Chunk Document

Step 4: Embedding Generation:

Load a pre-trained language model (e.g., sentence-transformers/all-MiniLM-L6-v2) using Hugging Face. This model converts each chunk into dense vector embeddings representing the chunk's semantic meaning. Apply the embedding model to generate vector representations for each text chunk. These embeddings make it easier to perform similarity searches based on the semantic content of each chunk.

Step 5: Store Embeddings in a Vector Database:

Once document chunks have been created and transformed into embeddings, these embeddings are stored in a vector database, such as Chromadb. The vector store allows for efficient similarity searches and fast retrieval of relevant data by indexing the embeddings in a structured, searchable format. Each chunk's embedding is stored along with a unique identifier (UUID) and relevant metadata, including the original document's title or source of the document.

```
vector_store.add_documents(documents=processed_documents, ids=uuids)

['afd9cb02-e1a2-48fe-bb56-eb920dcb3cfd',
 '786f6505-1dca-4245-8265-97dcc3df704f',
 'ea889b07-949e-418c-8903-df4682d104e7',
 'cbbcfea6-8644-4cb6-b59e-d6497a503e26',
```

Figure 3: Store Embeddings in a Vector Database

Step 6: Extract Unique Document Sources and Perform Similarity Search for Source:

In this step, the system identifies unique document sources by extracting metadata from each processed chunk, such as filenames or document titles, to create a list of distinct sources. This helps in organizing the retrieval process by document origin. For each source, the system uses the user's query to perform a similarity search within the vector database, retrieving the two most relevant chunks per document.. These two chunks from each source are then combined to create an information-dense block for that source. This step ensures that the system can access the most relevant sections from each document, providing a focused response to the user's query.

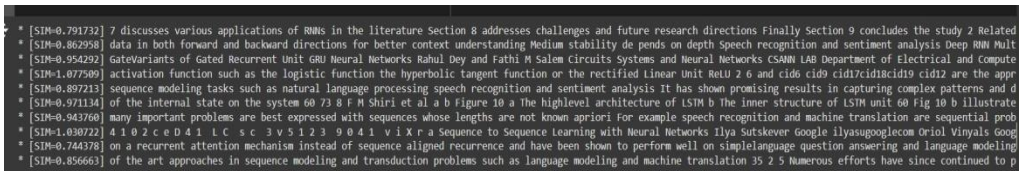

Figure 4: Retrieved Document

Step 7: Generate Source-Specific Answers and Final Response:

In this step, the system uses the LLaMA3 (inference taken using the Groq API) model to process the retrieved chunks for each source. Each set of combined chunks for a specific research paper is passed to the model, along with the user's query, to generate an answer specific to that source. The LLaMA3 model provides an answer for each document, ensuring responses are tailored to the content of individual sources.

Once the source-specific answers are generated, they are combined into a unified context. This combined context, containing the answers for each source, is then fed back to the LLaMA3 model. The model generates a final answer that synthesizes the information from all sources, offering a comprehensive and cohesive response to the user's query. This approach enhances the accuracy and depth of the final answer by incorporating insights from each relevant document.

Step 8: Save and Display the Final Answer:

In this final step, the system saves the generated answer in a structured format to provide a clear, well-organized response to the user's query. The comprehensive answer, which integrates insights from all processed sources, is stored in a text file, ensuring it is easily accessible for future reference or further analysis.

Result:

Query:

Figure 5: Question asked by User

Answer:

Figure 6: Answer Given by RAG System

Conclusion:

There are many ways to create a question-answering system, and each approach has its own drawbacks and benefits. Retrieval-Augmented Generation (RAG) is one such method through which we can generate precise answers in a question-answering system. It's actually hybrid method means a combination of Extractive and Abstractive [3]. RAG is one of these methods, combining both extractive and abstractive techniques for improved performance. In RAG, information retrieval (via ChromaDB) fetches the relevant chunks, and a generative model (like LLaMA3) synthesizes these chunks to produce a coherent, natural answer. This hybrid approach ensures the final answer is contextually rich, more accurate, and aligned with the user's query.

References:

1. James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.
2. Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.
3. Bhagat Gayval and Vanita Mhaske. Evaluation of descriptive probability approach, cosi pretrained mo. *Journal ofScientific Research*, 67(2), 2023.
4. M Lewis. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

5. Ibomoiye Domor Mienye, Theo G Swart, and George Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9):517, 2024.
6. Payal Mittal. A comprehensive survey of deep learning-based lightweight object detection models for edge devices. *Artificial Intelligence Review*, 57(9):242, 2024.
7. Dinidu Sandaruwan, Subha Fernando, and Sagara Sumathipala. Neural machine translation approach for singlish to english translation. *The International Journal on Advances in ICT for Emerging Regions*, 14(03):36–42, 2021.
8. Farhad Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru. *ArXiv*, abs/2305.17473, 2023.
9. I Sutskever. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.
10. A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
11. Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.