**Intro:** Recent advancements in retrieval-augmented generation (RAG) systems represent a significant development in natural language processing, combining the powerful generative capabilities of language models with external information retrieval mechanisms to enhance factual accuracy and contextual relevance. The core architecture of RAG systems integrates two primary components: a retriever that leverages both dense and sparse information retrieval techniques to source external documents, and a generator that incorporates this information into its responses. This integration allows the model to augment original prompts with data not necessarily stored within the model itself, thereby generating more accurate and contextually appropriate outputs. Such an approach is particularly valuable when dealing with unique datasets not encompassed in a model's training corpus, presenting opportunities to develop customized tools under specific data constraints. We aim to develop an advanced natural language query interface for data from the Library of Congress, integrating web scraping and data structuring techniques to enable intuitive and efficient information retrieval. By enhancing user interaction, increasing data accessibility, and streamlining information discovery, our research assistant enables scholars, investigators, journalists, and the general public to readily access the specialized and diverse information repositories housed at the Library of Congress.

**Model:** We designed an advanced Retrieval-Augmented Generation (RAG) system is a comprehensive pipeline designed for generating contextually accurate answers to user queries seeking insights into [manuscripts, etc] that are stored in the Library of Congress. Several modules for text preprocessing, chunking, retrieval of relevant information, and response generation work together to provide relevant outputs that accelerate research and help fill knowledge gaps/increase understanding. Preprocessing and chunking of textual data uses MarkdownHeaderTextSplitter and RecursiveCharacterTextSplitter from the LangChain library to split documents based on structural elements such as headers and character count, respectively. The preprocessing removes irrelevant text and cleans up formatting to ensure the input data is optimized for downstream tasks.

The retrieval functionality is responsible for embedding generation and vector search operations. The retriever function uses the instructor-xl sentence transformer, designed for producing high-quality sentence embeddings. The encoded chunks of text are indexed using FAISS for efficient similarity search, which allows for scalable and rapid querying of textual datasets. The state-of-the-art nature of the embedding models ensures that the retrieval process is accurate suitable for a wide array of topics.

Text responses are generated by providing contextually relevant documents and prompts to the relatively compact yet powerful T5-small model. The generator takes the most relevant passage from the retriever and crafts a response that directly addresses a user’s query for whatever they are researching. The selection of flan-t5-small, a relatively compact yet powerful model, balances computational efficiency with output quality, making the RAG system suitable for real-time applications where latency is a concern.

Type of RAG…

* Naïve:
* Advanced: adds pre-retrieval and post-retrieval process to Naïve RAG to improve the quality of the responses
* Modular: introduces enhanced functionalities to the Naïve RAG; integrates a search module for similarity retrieval and adopts a fine tuning approach in the retriever
  + Hybrid
  + Recursive retrieval and query engine
  + Stepback approach
  + Sub queries
  + Hypothetical document embeddings

Our Approach to RAG

* Supplements to Naïve RAG type
* Retriever strategy
* Generator strategy
* Model decisions
* Data retrieval
  + GROBID for PDFs (??)[[1]](#footnote-1)[[2]](#footnote-2)

Question generator[[3]](#footnote-3)

* Apply NER on each doc
* Use the most prominent subject in each document as an ‘answer’
* Follow the cited git to get questions for the supplied answer and doc
* Repeat for each document

Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks[[4]](#footnote-4)

* By decomposing complex RAG systems into independent modules and specialized operators, it facilitates a highly reconfigurable framework. Modular RAG transcends the traditional linear architecture, embracing a more advanced design that integrates routing, scheduling, and fusion mechanisms.

The Power of Noise: Redefining Retrieval for RAG Systems[[5]](#footnote-5)

* RAG is primarily designed to improve factual accuracy by providing the model access to auxiliary information, thereby augmenting the original prompt with information not necessarily memorized in the LLM. A key benefit of this approach is that it helps ground the prompt with relevant information that might help the LLM generate more accurate answers at inference time. At their core, RAG systems consist of two fundamental components: a retriever and a generator. The retriever is responsible for invoking an external IR system (dense and/or sparse) and feeding the selected results to a generator component.
* We argue here that the retrieval component of RAG systems, be it dense or sparse, deserves increased attention from the research community, and accordingly, we conduct the first comprehensive and systematic examination of the retrieval strategy of RAG systems. We focus, in particular, on the type of passages IR systems within a RAG solution should retrieve. Our analysis considers multiple factors, such as the relevance of the passages included in the prompt context, their position, and their number.
* One counter-intuitive finding of this work is that the retriever's highest-scoring documents that are not directly relevant to the query (e.g., do not contain the answer) negatively impact the effectiveness of the LLM. Even more surprising, we discovered that adding random documents in the prompt improves the LLM accuracy by up to 35%.

RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture[[6]](#footnote-6)

* Our pipeline consists of multiple stages, including extracting information from PDFs, generating questions and answers, using them for fine-tuning, and leveraging GPT-4 for evaluating the results.
* Considering this, we employed GROBID[[7]](#footnote-7) (GeneRation Of BIbliographic Data) (GRO, 2008–2023), a machine learning library specifically tailored for extracting and processing data from scientific literature in PDF format. The goal is to transform unstructured PDF data into structured data in the form of TEI (Text Encoding Initiative) format (Consortium, 2023), efficiently managing large volumes of files. The use of GROBID, trained on a vast corpus of scientific articles, enables the recognition of a wide array of document elements and extraction of associated bibliographic data.
* We aim to generate contextually grounded and high-quality questions that accurately reflect the content of the extracted text. For this, we employ the Guidance framework (Gui, 2023), whose primary advantage lies in its capacity to provide unparalleled control over the structural composition of both inputs and outputs, thereby augmenting the overall efficacy of response generation from language models. This degree of control results in outputs that are not only more precise, but also exhibit enhanced coherence and contextual relevance.
* Embedding generation and index construction: we compute embeddings from text chunks extracted from the PDF documents in our dataset, using sentence transformers (Reimers and Gurevych, 2019). We then used Facebook AI Similarity Search (FAISS) (Johnson et al., 2019), a library for efficient indexing and similarity search of vectors, to create a database of the embeddings…Specifically, we provided the retrieved information from the FAISS database to GPT-4 as context within a custom prompt, which allowed the generation of domain-specific answers.

Good overall graphic of RAG system[[8]](#footnote-8)

1. https://arxiv.org/abs/2401.08406 [↑](#footnote-ref-1)
2. https://medium.com/@researchgraph/how-to-use-grobid-67df995b16fa [↑](#footnote-ref-2)
3. https://github.com/MohammedAly22/GenQuest-RAG [↑](#footnote-ref-3)
4. https://arxiv.org/abs/2407.21059 [↑](#footnote-ref-4)
5. https://dl.acm.org/doi/pdf/10.1145/3626772.3657834 [↑](#footnote-ref-5)
6. https://arxiv.org/abs/2401.08406 [↑](#footnote-ref-6)
7. https://medium.com/@researchgraph/how-to-use-grobid-67df995b16fa [↑](#footnote-ref-7)
8. https://medium.com/enterprise-rag/an-introduction-to-rag-and-simple-complex-rag-9c3aa9bd017b [↑](#footnote-ref-8)