**Abstract (pending)**

Recent advancements in Retrieval-Augmented Generation (RAG) systems represent a significant development in natural language processing, combining the generative capabilities of language models with external information retrieval mechanisms to enhance factual accuracy and contextual relevance. They offer a particularly valuable opportunity to develop tools with private and proprietary information or data that may not have been included in the model’s training corpus. We aim to develop a 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.

**Introduction**

RAG systems represent a cutting-edge solution in natural language processing, combining the power of state-of-art large language models (LLMs) with targeted information retrieval. By supplementing LLMs with external data sources stored in a vector database, RAG systems can overcome some of the risks of hallucinations and the static nature of training data. The core components of RAG involve a retriever that locates and collects the necessary information from the vector database and a generator that incorporates this information to produce contextually rich and accurate responses. The retrieval component can operate using either dense or sparse vector spaces to ensure relevance and accuracy in the data retrieved, with the former capturing more complex semantic relationships between words and phrases that enable a more nuanced understanding and retrieval of information.[[1]](#endnote-1) The generator component then is an LLM that adapts its responses based on the information provided by the retriever.[[2]](#endnote-2) By implementing and evaluating different retrieval and generation strategies, the output returned to the user in response to a query can be drastically improved.

Archival collections, particularly those housed online, can present significant challenges to researchers due to the fragmented nature of their metadata and descriptive systems. A single collection may be described in multiple ways using differing metadata schemas stored in alternate locations that might not be clearly connected. For example, the Library of Congress (LoC) collections, and specifically those of the American Folklife Center (AFC), have documents, transcripts, recordings, and more, all relating to each other in different ways and interconnecting through different avenues. A typical collection might have a catalog record in catalog.loc.gov, a finding aid in findingaids.loc.gov, and digital surrogates (e.g., images, texts, audio, or video) in loc.gov/collections—all of which may or may not clearly link to each other. This lack of cohesive connectivity can leave researchers, staff, or both confused or searching for an extending period of time as they try to cope with partial recollections or vague leads. For a more naturally engaging experience with collections like these, this project seeks to design an interactive system that provides a more informative experience while improving research efficiency and precision.

**Approach**

This project centered on collections from the American Folklife Center (AFC), established in 1976 (WHY?). These holdings encompass a diverse array of analog and digital materials, including photographs, oral history recordings, folksongs, correspondence, and manuscripts such as field notes and recording/photo logs, along with broader archival collections. The collected data is transformed into multiple vector stores through various embedding strategies and models, each integrated into different RAG architectures.

A basic naïve RAG retrieval as well as more advanced strategies were constructed, and different configurations are tested and evaluated for retrieval accuracy through a two-phase process. Initially, a small data sample is used to narrow down the most promising RAG configurations. Additionally, our naïve RAG and first advanced RAG system were used during this phase. Hypothetical document embedding (HyDE) strategies have shown promise at significantly improving document retrieval. Our first advanced RAG system implements a basic HyDE mechanism ahead of the retriever. Evaluation revealed two optimal chunking strategies and two embedding models for further analysis. Second, the entire dataset was employed to identify the optimal retrieval process. Three vector stores were constructed using the approaches identified during the initial tests. The same RAG architectures were also compared, in addition to a third advanced system. The HyDE mechanism was enhanced, and a re-ranking module was incorporated to reassess the relevance of the retrieved documents.

This approach not only enhances the accessibility of archival collections but also affirms the performances of retrieval strategies on vector stores constructed using complex data structures. It empowers researchers to reliably engage with these systems using natural, intuitive queries, bridging the gaps between technical metadata, human inquiry, and comprehensive information.

**Data Gathering (Paul)**

Web scraping techniques and the loc.gov API provided these records for all AFC collections, including metadata that enabled the downloading of relevant PDF and audio files; non-text files were processed via optical character recognition (OCR) and machine transcription methods, for a total of 45,050 documents and corresponding metadata records. This represents the entirety of AFC’s publicly available digital collections, as of October 2024.

With the data aggregated, different chunking strategies and embedding models were used to generate the RAG system’s vector store. Initial examination on a small subset revealed….

**Vector store (Paul)**

After parsing and processing the data, our documents are broken up into several vector stores for evaluation.

…chunks of characters using Langchain’s *RecursiveCharacterTextSplitter*. A 15% overlap between the chunks is applied to maintain context and help improve retrieval accuracy. We used Instructor-XL as the default embeddings model (or MiniLM) and Deeplake vectorstore to house the embedded data.

Five primary vector stores were retained for the final application, two following our testing on the subset of the data and three used for the entire dataset.

* Subset of the data embedded into chunks of 250 characters, encoded by hkunlp/instructor-xl.
* Subset of the data embedded into chunks of 1,000 characters, encoded by hkunlp/instructor-xl.
* All of the data embedded into chunks of 250 characters, encoded by hkunlp/instructor-xl.
* All of the data embedded into chunks of 1,000 characters, encoded by hkunlp/instructor-xl.
* All of data embedded into chunks of 250 characters, encoded by amazon.titan-embed-text-v2:0.
* All of data embedded into chunks of 1,000 characters, encoded by amazon.titan-embed-text-v2:0.

**Architectures**

**Builds:** To develop an optimal RAG system, three architectures were developed and implemented, growing from a naïve RAG system to incorporating advanced methods, like hypothetical document embedding strategies and re-ranking. In all cases, the LLM used in this project is anthropic.claude-3-5-sonnet-20240620-v1:0. Depending on the user’s preference and for comparing purposes, two embedding models are primarily used throughout: hkunlp/instructor-xl (public) and amazon.titan-embed-text-v2:0 (private). A third embedding model is briefly used in early sample testing but soon after dropped as a consideration due to poor performance: all-MiniLM-L6-v2 (public).

* **Naïve RAG:** The naïve RAG architecture receives the query from the user, embeds it using the same model that was used to construct the vector store (titan or instructor), and sends the top\_k documents relevant to the query to the LLM for response generation.

A diagram of a software company

Description automatically generated

* **RAG with Hypothetical Document Embeddings (HyDE):** A more advanced RAG architecture includes a HyDE component that aims to provide the retriever with additional context to improve retrieval accuracy. The user’s query is supplied to an LLM that generates a hypothetical response. The query and response is combined to form a hypothetical document that gets embedded and searched in the vector store. The top\_k matching documents are then provided back to the LLM for response generation.

A close-up of a computer screen

Description automatically generated

* **RAG with Advanced Hypothetical Document Embeddings and Re-ranking (HyDER):** The final architecture increases the complexity of the HyDE component and adds a re-ranking feature at the end to prioritize the best scoring documents and reduce noise.The hypothetical document generator from the previous HyDE architecture is expanded to produce two additional hypothetical documents. The LLM receives the user’s query and generates two derivative queries and responses to all three for three hypothetical documents. Each are embedded and passed to the vector store that retrieves top\_k matching documents for *each* hypothetical document. All of the results are combined and de-duplicated and then passed to a re-ranking mechanism. The pooled documents are re-ordered according to their final relevancy scores and the best scoring top\_k documents are retained and sent to the generator to produce a response for the user.

A screenshot of a computer

Description automatically generated

**Hypothetical Document Embeddings Methodologies:** The hypothetical document generator is integrated early in the system using an LLM to generate and process variations of the users' query and their responses.The LLM generates an initial response for the original query, rewrites the query into slightly different variations, and generates answers for each variation. For the HyDER architecture, each of the three query-and-responses are generated using progressively higher temperatures (i.e., 0.7, 0.8, 0.9) to preserve factual integrity but also ensuring a degree of variety; the HyDE system applies a 0.7 temperature. All of the generated content is passed through a cleaner to compensate for instances when the LLM does not generate adequate output, or if the output includes language not relevant to the objective. For example in some cases, the LLM refrains from forming a response out of its perceived copyright concerns. In such instances, valid output is taken from the other hypothetical documents to assure each hypothetical document is complete for passing forward.

* Derivative query generation prompt:

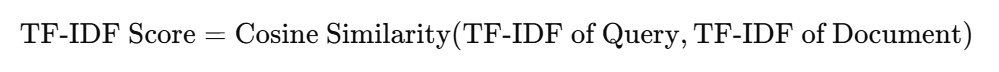
*Rewrite this query to be slightly different but similar in meaning: {query}*

* Hypothetical response generation prompt:

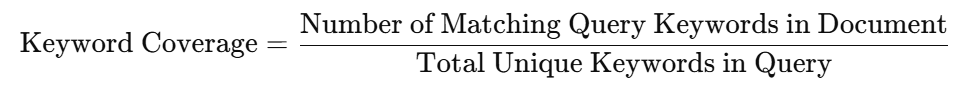
*You are a document that answers this question {query}.*

*Write a short, natural paragraph that directly answers this question. Include additional relevant information if possible.*

**Re-Ranker Details:** The re-ranking function employs a multi-faceted approach to assess and rank documents based on their relevance to a query. The process integrates *sklearns* TF-IDF scores, a basic keyword coverage calculation, and document freshness based on its date to achieve an effective ranking mechanism. The TF-IDF portion assesses the relevance of documents’ content to a query. Vectorization tokenizes and transforms the text of all documents, as well as the query, into a matrix of TF-IDF features. Words that are frequent in a single document but rare across the corpus are given higher scores. Cosine similarity is calculated between the query's TF-IDF vector and each document's TF-IDF vector to get a relevance score for each document. A higher score for a document indicates a greater alignment with the query's content.



To further refine relevance, the function calculates keyword coverage by determining the proportion of unique words in the query that are explicitly present in each document's text. For each document, the number of unique query words found in the document is divided by the total number of unique words in the query. This calculation results in a keyword coverage score, which rewards documents that include a higher proportion of the query terms, emphasizing direct textual matches.

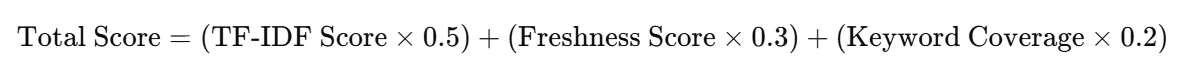


To prioritize recent documents, a freshness score is computed based on the age of the document relative to the current date. Each document’s metadata is checked for a timestamp and, if available, assigned a score based on its proximity to the present day, with newer documents receiving a higher score.

A black text on a white background

Description automatically generated

The final relevancy score integrates the TF-IDF and similarity calculations with the recency and keyword coverages scores to determine which documents are most related to the input query. Different weights are assigned to each to emphasize TF-IDF and cosine similarity. Based on the total score, documents are re-ordered and the best scoring top\_k documents are retained for use in response generation.



**Generator:** The top\_k retrieved documents each contain text that gets combined into a single ‘context’ variable. For token length efficiency and to avoid mixing context with unmatching metadata, only the best scoring document’s metadata is used in generating a response to the user’s original query.

*Human: Please answer the following query based on the provided context and metadata.*

*Query: What are the key findings from the 2024 climate study?*

*Context: [Relevant text from top-ranked documents]*

*Metadata: [Relevant metadata from top-ranked document]*

*Instructions:*

*1. Answer the question using ONLY the information provided in the Context and Metadata above.*

*2. Do NOT include any information that is not explicitly stated in the Context or Metadata.*

*3. Begin your answer with a direct response to the question asked.*

*4. Include relevant details from the Context and Metadata to support your answer.*

*5. Pay special attention to the recording date, contributors, and locations provided in the metadata.*

*6. Inform the user of what document filename they can find the information in.*

*Your Answer here:*

**Evaluation:**

**Phase I – Narrowing retrieval strategies:** A small sample of data was extracted and used for initial testing. Vector stores built in chunks of 250, 500, 1,000, and 2,000 were examined using the three different models. Naïve and HyDE architectures were also compared to gauge the efficacy of HyDE as a RAG strategy. Each system configuration was tested with 100 questions, each accompanied by the unique filename of the document that contained the question’s correct answer. Accuracy was measured on whether the correct document was included in the top\_k filenames returned by the retriever.

As expected, all-MiniLM-L6-v2 as an embedder and the Naïve architecture proved inadequate. The basic HyDE architecture showed significant improvements in retrieval accuracy, generally increasing it between ten and twenty percentage points in most system configurations. Surprisingly, the vector store constructed using embedding chunk sizes of 250 characters, a particularly short length of text, competed with that of the same using 1,000 sized chunks for the optimal vector store. The most promising system architectures consisted of the basic HyDE generator, and instructor-xl or titan as the embedding model to construct the vector stores in chunk sizes of 250 or 1,000. In testing on the sample set, 90% retrieval accuracy on the 100 sample test questions was achieved by within a top\_k of four documents.

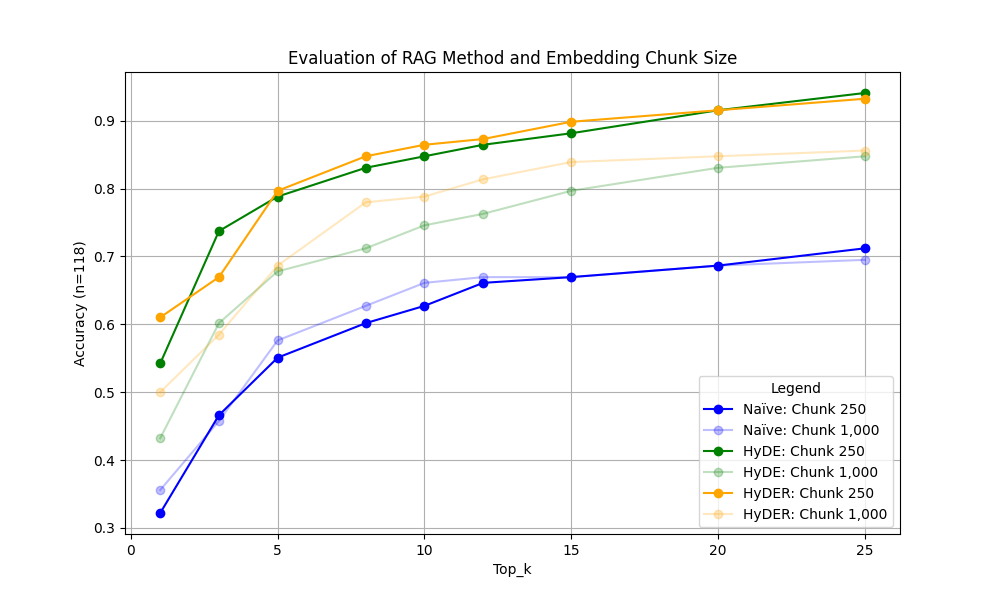
A group of graphs with different colored lines

Description automatically generated

**Phase II – Applying to all data:** Evaluation of the RAG system’s configurations was expanded to the entire dataset. Vector stores based on 250 and 1,000 chunking strategies consisted of 1.6 million and 400,000 million chunked documents, respectively. Similar to the sample data, 118 new questions were used to gauge retrieval accuracy. The same naïve and HyDE architectures were again compared, as well as HyDER now being included.

Although both instructor-xl or titan embedding models showed promising results, sample accuracies were compared between the two to identify which provided better retrieval results. For efficiency and considering computation and time constraints, only the better performing one was applied in all system configurations. Under the HyDER architecture, separate tests with top\_k values of one and three on vector stores chunked by 250-character blocks showed noticeably different retrieval accuracies. The retriever mechanism with titan-generated embeddings showed accuracies of 45% and 59% for top\_k values of one and three, respectively, while the instructor-generated embeddings showed accuracies of 61% and 67% for the same top\_k values. Therefore, titan was dropped and all RAG systems and components were compared using the instructor-xl model as the embedder.

Final evaluations showed close performances between the HyDE and HyDER architectures, with HyDER narrowly performing better. Additionally, similar to our findings on the subset of the data, the vector store constructed using 250-character sized chunks provided better results. Both the architectures on 250 chunk-sized vector stores achieve 90% accuracy at or just beyond a top\_k value of 15, and plateau at around 94% after top\_k of 25.



**Final Insights:** The close performances between the HyDE and HyDER systems suggest that the significantly more complex HyDER may not be necessary to achieve optimal results. Basic HyDE mechanisms in most instances are likely sufficient to drastically improve retrieval accuracies.

1. Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The Power of Noise: Redefining Retrieval for RAG Systems. In Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24). Association for Computing Machinery, New York, NY, USA, 719–729. <https://dl.acm.org/doi/abs/10.1145/3626772.3657834> [↑](#endnote-ref-1)
2. Wu, S., Xiong, Y., Cui, Y., Wu, H., Chen, C., Yuan, Y., ... & Xue, C. J. (2024). Retrieval-Augmented Generation for Natural Language Processing: A Survey. arXiv preprint arXiv:2407.13193. <https://arxiv.org/abs/2407.13193> [↑](#endnote-ref-2)