**FolkRAG: A Retrieval-Augmented Generation System for Cultural Heritage Materials**

**Abstract**

Retrieval-Augmented Generation (RAG) systems represent a cutting-edge innovation in natural language processing, combining generative language models with external information retrieval to enhance accuracy and contextual relevance. This study introduces *FolkRAG*, a proof-of-concept system designed to query public data from the American Folklife Center (AFC) at the Library of Congress, addressing challenges posed by fragmented and inconsistent metadata across archival collections. By optimizing vector store parameters and implementing advanced RAG retrieval strategies using hypothetical document embeddings and re-ranking, *FolkRAG* demonstrates the potential to improve access to cultural heritage materials, enabling natural language queries and coherent, citation-supported responses while upholding the core tenets of librarianship.

## Introduction

A single archival collection can be, and often is, described and presented online in a variety of ways that incorporate different metadata and metadata schema housed in different locations that, while connected intellectually, may or may not be connected semantically. Elements of that collection are often displayed online in the form of digital surrogates that utilize additional technical and descriptive information that, again, may or may not link back to a central document that allows a researcher to make sense of the material.

The collections at Library of Congress (LoC), and specifically the digital representations of those of the American Folklife Center (AFC) upon which this experiment centers, adhere to this. A collection is typically represented by a catalog record which may or may not link to a finding aid in on LoC's website, if it has one; the finding aid may or may not link back to the catalog record; there may be digital versions of objects from the collection (image, text, audio, or video) in loc.gov/collections that are sometimes, but not always, linked from the catalog record or finding aid; those objects have metadata that mostly, but not uniformly, link back to the previously mentioned sources.

It is impossible to search all of these data sources at once. A researcher must rely on their own facility with archival research to successfully navigate a collection, and, when necessary, on the guidance of reference librarians who sometimes know these collections intimately, but often do not. Researchers and staff alike rely on these systems of description to guide them. There is also no way to interact with these discovery systems in a way that accounts for half-remembered details, threads upon which an individual can pull for a more natural or esoteric entry point into a subject area. One cannot ask questions without in-depth knowledge of how these systems work, which few people truly possess. The aim of this project, therefore, is to build a system that addresses these concerns computationally without abandoning the core tenets of librarianship - service, accountability, context, and authority.

RAG represents a significant advancement in natural language processing, combining large language models (LLMs) with targeted information retrieval capabilities. At their foundation, RAG systems supplement LLMs with external data sources stored in vector databases, enabling them to overcome limitations like hallucinations and static training data (Veturi et al., 2024). The core architecture comprises a retriever that locates relevant information from the vector database and a generator that incorporates this information to produce accurate, contextually-rich responses.

The retrieval component can operate in either dense or sparse vector spaces, with dense embeddings capturing more nuanced semantic relationships between words and phrases. The choice of embedding model significantly impacts system performance. For instance, Instructor-XL offers superior understanding of nuanced language but requires substantial computational resources, while AWS Bedrock's Titan model enables faster processing at scale despite potential challenges with high-volume API calls.

The evolution of RAG systems has progressed through three distinct paradigms, as outlined by Gao et al. (2023). The initial Naive RAG paradigm, introduced by Lewis et al. (2020), established the foundational "Retrieve-Read" framework. While groundbreaking, this approach faced limitations in retrieval precision and potential hallucination issues when handling irrelevant information, as demonstrated by Borgeaud et al. (2022) in their work with large-scale document collections.

Advanced RAG represents a significant evolution, introducing sophisticated optimization strategies. A key innovation is Sentence Window Retrieval, which decouples embedding and synthesis processes. Yang (2023) demonstrated how this technique uses smaller text units for retrieval while maintaining broader context for generation. Another significant advancement is Hypothetical Document Embedding (HyDE), developed by Gao et al. (2022), which generates hypothetical answers using LLMs to improve retrieval accuracy through answer-to-answer comparison. Pal (2023) further validated this approach's effectiveness in reducing hallucinations.

The reranking component of Advanced RAG has seen substantial development, as detailed by Bhavsar (2023). Various approaches including cross-encoders, multi-vector models, and LLM-based rerankers have been implemented to improve retrieved document relevance. Notably, Cuconasu et al. (2024) demonstrated that random or noisy documents, when strategically positioned in the context, can sometimes improve rather than degrade LLM performance.

## Proposed Solution

### **Data Collection and Metadata**

AFC, founded in 1976 when Congress passed the American Folklife Preservation Act, has fulfilled its charge to preserve and present folklife in all its diversity, and documents and shares the many expressions of human experience to inspire, revitalize, and perpetuate living cultural traditions. It houses the Archive of Folk Culture, which was originally founded as the Archive of American Folk Song in 1928. The holdings comprise documentation of traditional culture from around the world and feature multiformat materials in both analog and digital formats.

The catalog is typically the first place a public record of a library resource is created. It contains high level bibliographic information about an item, and allows that information, whether it pertains to a book, archival collection, or other format, to be located. The record is created by a cataloger using the Machine Readable Cataloging (MARC) format and is available as MARC XML. A bibliographic item is typically represented by a single catalog record.

Archival finding aids describe, in a broad sense, what a collection contains - if physical, its container list will document each box, folder, and sometimes item; if born-digital, its container list will document whatever arrangement has been imposed on the digital materials; if both, then a combination of each approach is used. Finding aids also provide contextual information about the record creators, related materials, provenance, and how the collection has been arranged, why, and by whom. They also utilize controlled vocabularies such as subject headings to provide researchers with a general impression of the topics covered in the collection. They take the form of structured XML known as Encoded Archival Description (EAD). An archival collection will typically be represented by a single EAD, written by an archivist once a collection has been processed and is available for research.

For our project, we wrote a web scraper to obtain individual EAD XML and MARC XML records for all AFC collections and utilized the LoC API to pull AFC digital collection JSON metadata at the file level and transform it into .csv. Since that metadata contains file locations and mime-types, we then structured and executed wget requests to download all AFC .txt, .pdf, .mp3, and .mp4 files. We extracted all .pdf text with tesseract optical character recognition (OCR) and transcribed (and, where appropriate, translated into English) all .mp3s with OpenAI's large Whisper speech recognition model. We scraped 158 EAD XML files, 158 MARC XML files, and obtained metadata for 48 digital collections. Our wget pulls followed by either OCR or Whisper transcription resulted in acquisition of 10,980 machine generated transcripts, 4,292 OCR files, and 29,778 library-created text files for a total of 45,050 documents and corresponding metadata records, representing the entirety of AFC's publicly available digital collections, as of October 2024.

Our LoC API queries resulted in two .csv files for each collection - search\_results.csv and file\_list.csv. Each request is based upon a search, and each search returns a list of resources and their descriptive metadata - this is returned as JSON and saved as a .csv. Each resource can contain any number of derivative files, and search\_results.csv contains links to the item pages where those files are available. This information is used in a second API call to the item pages to generate file\_list.csv - the list of every file for every resource in the search, its type, and its location.

The metadata processor takes a modular approach to integrating diverse archival description sources into a unified whole. It extracts and normalizes identifiers from URLs in various formats through pattern matching, generates potential matching patterns for different resource types, and implements thorough error handling and logging. For EAD and MARC, the processor parses hierarchical elements like title, date, abstract, subject headings, and series titles, and preserves the contextual information these sources provide at the collection level. The system maps related fields across schemas while maintaining source attribution, addressing the key challenge of disconnection between finding aids, catalog records, and digital objects.

The main processing function first parses the search\_results.csv to create multiple mapping dictionaries that associate different forms of identifiers (base IDs, digital IDs, and resource URLs) with their corresponding metadata. It then processes the file\_list.csv to establish direct file-to-identifier mappings. For each file encountered, the processor attempts multiple matching strategies: first trying an exact match through file\_list mappings, then attempting matches through the various identifier dictionaries if the direct approach fails.

The final metadata schema incorporates fields for identification (original\_filename, file\_type, chunk\_id, total\_chunks, call\_number), descriptive information (title, date, created\_published, language, type), authorship and custody (contributors, creator, repository, collection, source\_collection), content description (description, notes, subjects, original\_format, online\_formats), rights management (rights, access\_restricted), geographic coverage (locations), resource location (url), as well as specialized fields from finding aids and catalog records. Array fields like contributors, notes, and subjects are stored as JSON strings to preserve their multi-valued nature while maintaining compatibility with Deeplake's data structure.

### **Initial Approach and Vector Database Development**

In the initial development stages, we systematically explored multiple technical approaches while addressing increasingly complex requirements for processing archival materials. Our early experiments focused on identifying optimal solutions for embedding model selection and vector storage implementation. This collection encompassed a wide range of document types, presenting distinct processing requirements and metadata relationships that needed to be preserved throughout the retrieval process.

Our initial vector storage implementation revealed significant technical challenges that shaped the evolution of our system architecture. We began with Facebook AI Similarity Search (FAISS), attracted by its reputation for fast similarity search operations and efficient GPU support. Initial performance testing showed promising results, with FAISS demonstrating high-speed processing capabilities for collections ranging from 50,000 to 100,000 documents. However, as we progressed with implementing detailed archival metadata integration, FAISS's limitations became increasingly problematic. The system struggled to efficiently manage the complex hierarchical collection information, creator attribution details, and temporal metadata that are crucial in archival contexts.

These limitations led to a pivotal shift to Deeplake as our vector storage solution. Deeplake offered a more comprehensive framework for handling both document embeddings and complex metadata structures within a unified system. While this transition initially required additional development effort to migrate our existing implementation, it ultimately provided a more robust foundation for our RAG system. Deeplake's integrated approach eliminated the need for parallel metadata management systems, simplifying our architecture while maintaining the rich contextual information necessary for effective document retrieval.

This architectural pivot proved especially valuable as our collection grew beyond 100,000 documents. Deeplake's integrated approach to handling both embeddings and metadata enabled more sophisticated query operations that could simultaneously consider semantic similarity and archival context. The system demonstrated superior capability in maintaining the complex relationships between documents, their descriptions, and their place within larger archival hierarchies, while providing consistent query performance across our expanding collection.

The vector store utilizes DeepLake and LangChain as the underlying database technology, chosen for their ability to handle the scale and complexity of AFC's digital collections while maintaining efficient similarity search capabilities and metadata-based filtering. The system offers multiple embedding model options: HuggingFace's Instructor-XL, MiniLM, and Amazon's Titan embedding model. Each model presents different tradeoffs - Instructor-XL excels at understanding nuanced language and context but requires more computational resources, while Titan leverages AWS Bedrock's infrastructure for potentially faster processing at scale.

The vector store initialization process creates a comprehensive tensor structure that preserves both the textual content and its associated metadata. Each document is represented by a set of tensors including the raw text, embeddings, and all metadata fields established during processing. The system implements memory management through batch processing, with configurable batch sizes (defaulting to 100 documents) to handle the large volume of digital collection material efficiently.

Document chunking is performed using LangChain's recursive character text splitter with configurable chunk size and overlap percentage (defaulted to 15% overlap between chunks), preserving document boundaries and maintaining contextual coherence. For transcript files, the chunking process includes additional preprocessing to remove timecode annotations while preserving the temporal relationship between text segments.

The embedding generation process incorporates several safeguards to ensure data integrity:

1. A checkpointing system tracks progress and enables recovery from failures
2. Detailed logs of the embedding process track successful operations and document anomalies
3. Error handling at multiple levels, from individual document processing to batch operations
4. Memory optimization techniques including garbage collection and CUDA memory cache clearing

## Model Architectures

Our methodology centered on evaluating different RAG configurations through a two-phase process. Initially, a small data sample was used to narrow down the most promising RAG configurations. We implemented both a basic naïve RAG retrieval and an advanced system incorporating a hypothetical document embedding (HyDE) mechanism ahead of the retriever, as HyDE strategies have shown promise at significantly improving document retrieval. Evaluation revealed two optimal chunking strategies and two embedding models for further analysis.

Five primary vector stores were retained for the final application:

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

### **Builds**

To develop an optimal RAG system, three architectures were developed and implemented, expanding from a naïve RAG system to incorporating advanced methods, like hypothetical document embedding strategies and re-ranking. In all cases, the LLM used 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 Architecture**

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.

### **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 are 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.

### **RAG with Advanced HyDE 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 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 results are combined and de-duplicated before passing through 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.

### **Hypothetical Document Embeddings Methodologies**

The hypothetical document generator is integrated early in the system using an LLM to generate and process variations of users' queries 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 while ensuring variety; the HyDE system applies a 0.7 temperature.

All generated content passes through a cleaner to compensate for instances when the LLM does not generate adequate output or includes language not relevant to the objective. For example, in some cases, the LLM refrains from forming a response due to perceived copyright concerns. In such instances, valid output is taken from other hypothetical documents to ensure each hypothetical document is complete for passing forward.

The system employs two specific prompts:

*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 sklearn's TF-IDF scores, 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.

The complete re-ranking calculations are:

*Total Score = (TF-IDF Score x 0.5) + (Freshness Score x 0.3) + (Keyword Coverage x 0.2)*

*TF-IDF Score = Cosine Similarity (TF-IDF of Query, TF-IDF of Document)*

*Freshness Score = (0, 1 - Time Different in Days/365)*

*Keyword Coverage = Number of Matching Query Keywords in Document/Total Unique Keywords in Query*

### **Generator Component**

The generator combines the retrieved documents' text into a unified context variable and employs a structured prompt template for the LLM:

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

*Query: [User Query]*

*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.*

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.

## Results and Evaluation

### **Phase I - Narrowing Retrieval Strategies**

Initial testing employed a small data sample to evaluate different system configurations. Vector stores were built in chunks of 250, 500, 1,000, and 2,000 characters and examined using three different models. We tested both Naïve and HyDE architectures to gauge the efficacy of HyDE as a RAG strategy. Each system configuration was evaluated using 100 questions, with each question paired with the unique filename of the document containing its correct answer. Accuracy was measured based on whether the correct document appeared in the top\_k filenames returned by the retriever.

The results confirmed several key findings. The all-MiniLM-L6-v2 embedder and Naïve architecture demonstrated inadequate performance. In contrast, the basic HyDE architecture showed significant improvements in retrieval accuracy, generally increasing it between ten and twenty percentage points across most system configurations. Notably, the vector store constructed using embedding chunk sizes of 250 characters, despite being a particularly short length of text, performed comparably to the 1,000-sized chunks for optimal vector store performance. The most promising system architectures emerged as those combining the basic HyDE generator with either instructor-xl or titan as the embedding model, constructing vector stores in chunk sizes of 250 or 1,000. In testing on the sample set, these configurations achieved 90% retrieval accuracy on the 100 sample test questions within a top\_k of four documents.

### **Phase II - Full Dataset Implementation**

When expanding evaluation to the entire dataset, the vector stores based on 250 and 1,000 chunking strategies contained 1.6 million and 400,000 million chunked documents, respectively. We tested system performance using 118 new questions to gauge retrieval accuracy. The evaluation included comparisons of the Naïve and HyDE architectures, along with the newly introduced HyDER architecture.

While both instructor-xl and titan embedding models showed initial promise, we conducted detailed accuracy comparisons to determine the superior option. Under the HyDER architecture, testing with top\_k values of one and three on vector stores chunked by 250-character blocks revealed significant performance differences. The titan-generated embeddings achieved accuracies of 45% and 59% for top\_k values of one and three, respectively. In comparison, the instructor-generated embeddings demonstrated superior performance, achieving accuracies of 61% and 67% for the same top\_k values. Based on these results, we discontinued use of the titan embeddings and conducted all subsequent RAG system evaluations using the instructor-xl model as the embedder.

Final evaluations revealed close performance between the HyDE and HyDER architectures, with HyDER showing a slight edge. The vector store constructed using 250-character sized chunks consistently provided better results than larger chunk sizes. Both architectures achieved 90% accuracy at or just beyond a top\_k value of 15 when using 250 chunk-sized vector stores, with performance plateauing at approximately 94% after a top\_k of 25.

### **Detailed Example Analysis**

A thorough examination of specific query results provides clear evidence of the impact of chunk size on retrieval accuracy. Consider the following query: "What ingredients did the Vietnamese interviewee say go into pho?"

With top\_k=1, the results demonstrate the superior performance of the 250-character chunk size in retrieving the correct source material:

The 250-character chunks correctly retrieved the source document (mb\_r019\_01\_en.txt) with a similarity score of 0.8394821714969785. The retrieved text contained the actual interview response:

*They call it pho.*

*That's what they call it in my country, pho.*

*Ph-o.*

*Those are noodles?*

*Yes.*

*Noodles, they put some soup in there, some beef, some vegetables, something.*

*That's a famous food in my country.*

*Do you prepare that yourself at home?*

*Yeah.*

In contrast, the 1,000-character chunks retrieved an incorrect document (afc1987042\_mb\_r010\_01\_en.txt) with a similarity score of 0.8531693910577587. While this document discussed pho, it contained a different speaker's general description rather than the Vietnamese interviewee's direct response:

*Thin noodles.*

*Thin noodles.*

*Yeah.*

*You could do that on a thin yellow noodle.*

*Then it would be called bra mee leung lat na in the Thai language.*

*Very popular dish throughout all of China, all of Taiwan, Korea, Malaysia, Thailand, Laos, Cambodia, Vietnam, Burma is a thing called, the Vietnamese call it pho.*

*[...]*

*It's a beef broth soup.*

*Bean sprouts in it.*

*Thin rice noodles or wheat noodles.*

*Some fried garlic that we fry up ahead of time and have it sitting aside and just put it into the soup.*

*Would have scallions, coriander, the meat of your choice or shrimp and squid.*

When examining results with top\_k=5, we observed a pattern of duplicate retrieval:

250 Chunked vector store using hkunlp/instructor-xl:

* afc1987042\_mb\_r019\_01\_en.txt (retrieved 3 times, correct document)
* afc1987042\_mb\_r010\_01\_en.txt (retrieved 2 times)

1,000 Chunked vector store using hkunlp/instructor-xl:

* afc1987042\_mb\_r010\_01\_en.txt (retrieved 3 times, incorrect document)
* afc1987042\_mb\_r010\_r011.txt (retrieved 2 times)

### **Final Insights**

This detailed example analysis reveals several significant findings. The 250-character chunks proved more effective at pinpointing exact relevant content, even though the 1,000-character chunk retrieved text had a slightly higher similarity score. The phenomenon of the same document appearing multiple times in the results indicates that different chunks from the same source document were independently deemed relevant. While this redundancy might appear inefficient, the different chunks likely contained distinct contextual information that could be valuable for response generation. The 1,000-character chunks' tendency to retrieve thematically related but incorrect documents suggests that larger chunks may sometimes obscure specific relevant content within broader contextual information.

The close performances between the HyDE and HyDER systems suggest that the significantly more complex HyDER architecture may not be necessary to achieve optimal results. Basic HyDE mechanisms in most instances proved sufficient to drastically improve retrieval accuracies. However, our analysis of wrong retrieval samples revealed that the initial database construction or the de-duplication process might have been ineffective in some instances. The top\_k documents passed to the re-ranker sometimes sourced back to the same document filename, likely because single documents were broken up into chunks while retaining their source filename.

While the filenames may have been duplicative, the context associated with each instance would have been different, possibly mitigating any noticeable negative restraint on retrieval accuracies. Forcing the RAG system to identify unique documents equivalent to top\_k likely would have improved accuracy figures when top\_k was greater than one, but also would have compelled the system to send a number of documents to the generator that exceeded top\_k or would have dropped relevant context in favor of unique filenames. Alternatively, document IDs for each chunk could be generated to enable unique chunk identification. The full impact of this on accuracy metrics is not known at this time but warrants further scrutiny.

## Conclusion

FolkRAG demonstrates that RAG systems can effectively enhance access to complex archival collections while maintaining the integrity of archival description and context. The system's success in handling diverse document types and complex metadata relationships suggests promising applications for other cultural heritage institutions facing similar challenges in making their collections more accessible through natural language interaction.

The vector store constructed with 250-character chunks using the Instructor-XL embedding model provided the best balance of performance and efficiency. While both 250 and 1,000-character chunk sizes showed promise during initial testing, the smaller chunk size consistently delivered superior results when scaled to the full dataset. This finding suggests that finer granularity in text segmentation may better serve the nuanced nature of archival materials.

The implementation of comprehensive metadata processing proved crucial to the system's success. By preserving the complex relationships between documents, their descriptions, and their place within larger archival hierarchies, FolkRAG maintains the contextual integrity that is fundamental to archival research. This approach demonstrates that RAG systems can successfully bridge the gap between traditional archival description and modern natural language interaction while upholding professional standards of librarianship.

The potential issue of duplicate document retrieval warrants further investigation. While our system achieved high accuracy rates, the tendency to retrieve multiple chunks from the same source document could be addressed through improved de-duplication strategies or by implementing unique document identifiers for each chunk. This refinement could potentially improve both retrieval accuracy and the diversity of sources presented to users.

This research contributes to both the technical advancement of RAG systems and the practical application of AI in cultural heritage contexts. It demonstrates that careful attention to domain-specific requirements and metadata relationships can result in systems that not only improve access but also maintain the professional standards and contextual richness essential to archival research. Future work could explore the integration of multimodal content, enhanced metadata filtering capabilities, and more sophisticated re-ranking strategies that better account for archival hierarchies and relationships between collections.