Master

Enterprise Knowledge Management Through Caching Fine-Tuning

A Self-Improving Approach to Document Management and Information Retrieval

AHMAD ALKHALED

SUPERVISOR
MORTEN GOODWIN

University of Agder, 2024
Faculty of Engineering and Science
Department of Engineering and Sciences

Abstract

This paper is a pre-project for a master's thesis, aimed at providing a further understanding of innovative approaches to enterprise knowledge management.This thesis presents an innovative approach to optimizing Retrieval-Augmented Generation (RAG) systems in enterprise environments through the implementation of a "caching-fine tuning" process. The system addresses the significant challenge of managing large-scale document repositories by creating a self-improving mechanism that learns from user interactions. By collecting and leveraging actual employee queries and their responses, the system continuously refines its performance while reducing computational overhead. This approach not only improves response accuracy but also naturally identifies the most relevant documents within the organization's knowledge base. The implementation combines open-source Large Language Models (LLMs), specifically Ollama with Mistral, and modern RAG architectures to create a cost-effective, efficient, and continuously improving enterprise knowledge management system.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The exponential growth of digital documents within enterprise environments has created significant challenges in information management and retrieval [14]. Organizations today face the difficult task of managing millions of documents while ensuring quick and accurate access to relevant information [10]. Traditional document management systems, while functional, often struggle with the scale and complexity of modern enterprise data requirements.

The advent of Large Language Models (LLMs) has opened new possibilities in document management and information retrieval [10]. However, the computational resources required to process large document repositories and the challenge of maintaining accuracy in diverse business contexts remain significant obstacles [15].

1.2 Problem Statement

Organizations face several critical challenges in managing their document repositories [3]:

- **Scale:** Managing and processing millions of documents efficiently (for example, in this case, GE Healthcare Lindesnes handles 3.7 million local files, with additional files shared with other factories and departments such as Oslo and Cork through VeevaVolt).
- **Relevance:** Identifying truly important documents from vast repositories.
- **Resource Utilization:** Optimizing computational resources while maintaining system performance.
- **Accuracy:** Ensuring precise information retrieval and response generation.
- **Adaptation:** Keeping the system current with evolving business needs.

1.3 Research Objectives

This research aims to:

1. Develop an efficient RAG system that can handle large-scale document repositories
2. Implement a self-improving mechanism through user interaction
3. Optimize resource utilization by focusing on relevant documents
4. Create a scalable and maintainable enterprise knowledge management solution
5. Evaluate the effectiveness of open-source LLMs in enterprise settings.

1.4 Significance of the Study

This research contributes to the field of enterprise knowledge management by:

1. Introducing a novel approach to document relevance determination
2. Demonstrating the practical application of open-source LLMs in enterprise settings
3. Providing a framework for self-improving knowledge management systems
4. Addressing the critical challenge of resource optimization in large-scale systems

Chapter 2

Literature Review

2.1 Evolution of Enterprise Knowledge Management

Enterprise knowledge management has evolved significantly over the past dacades. Traditional document management systems focused primarily on storage and basic retrieval capabilities [2]. The introduction of semantic search and natural language processing marked a significant advancement, enabling more intuitive information retrieval [9].

Recent developments in artificial intelligence, particularly in natural language processing, have revolutionized how organizations approach knowledge management. The emergence of transformer-based models has enabled more sophisticated understanding and retrieval of information [1].

2.2 Large Language Models in Enterprise

Large Language Models have demonstrated remarkable capabilities in understanding and generating human-like text. Their application in enterprise settings has shown promising results in various areas [18] [6]:

1. Document Classification and Categorization
2. Information Extraction and Summarization
3. Question Answering Systems
4. Content Generation and Analysis

However, the deployment of LLMs in enterprise environments presents unique challenges [17]:

1. Cost of Implementation
2. Computing Resource Requirements
3. Privacy and Security Concerns
4. Integration with Existing Systems
5. Model Accuracy and Reliability

2.3 Open Source LLMs

The landscape of open-source large language models (LLMs) has grown quickly [18] [6], offering powerful alternatives to proprietary models [12] [11]. Several key open-source LLMs have stood out for their performance and versatility, making them popular choices for developers and businesses alike. Notable developments include:

2.3.1 Llama 2

Meta's release of Llama 2 marked a significant milestone in open-source LLMs, providing [16] [5]:

- Strong performance metrics comparable to proprietary models
- Various model sizes (7B, 13B, 70B parameters)
- Commercial usage rights
- Active community development

2.3.2 Mistral

Mistral is a highly efficient open-source model known for its [7]:

- Excellent performance-to-size ratio
- Optimized inference speed
- Strong reasoning capabilities
- Lower resource requirements

2.3.3 Ollama

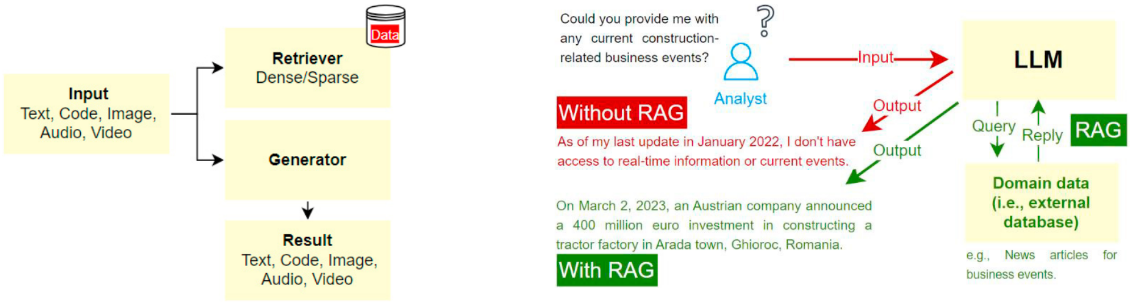
Ollama is an open-source tool that allows runing large language models (LLMs) directly on a local machine. This is especially useful for developers, researchers, and businesses who want to maintain full control over their data, ensuring privacy and security. By running models locally, Ollama helps avoid the potential risks of cloud storage and improves speed by reducing reliance on external servers [4].

With its easy-to-use setup, Ollama make it easy to manage various LLMs, customize them for specific tasks, and integrate them into projects. All without needing an internet connection. Whether building chatbots, conducting research, or developing AI-powered applications, Ollama provides a reliable and flexible solution for AI needs [4].

2.4 RAG Systems and Architectures

Retrieval-Augmented Generation (RAG) is a method used to improve the accuracy and relevance of text generated by machine learning models, especially Large Language Models (LLMs). While LLMs can generate impressive human-like text, they can struggle with specific, domain-related queries, sometimes producing incorrect or irrelevant information [8].

RAG addresses this issue by combining two key steps: first, the model retrieves relevant information from external data sources, such as databases or documents; then, it uses this retrieved information to generate more accurate and contextually appropriate responses. This integration ensures that the generated text is based on up-to-date and relevant data, reducing the likelihood of errors and making the model's outputs more reliable for real-world applications, such as business intelligence, medical diagnosis, or question-answering tasks [8].



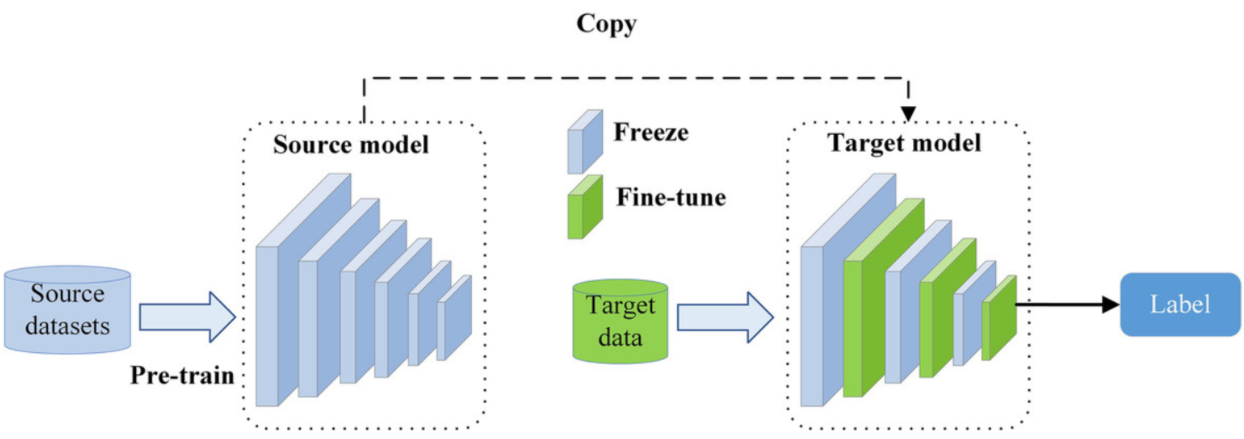
(a) (b)

Figure 2.1: (a) A generic RAG architecture, where users' queries, potentially in different modalities (e.g., text, code, image, etc.), are inputted into both the retriever and the generator. The retriever scans for relevant data sources in storage, while the generator engages with the retrieval outcomes, ultimately generating results across various modalities; (b) Illustrates how RAG integration with the LLM handles queries that fall outside the scope of the LLM's training data.

2.5 Fine-tuning Approaches

Fine-tuning is a machine learning technique used in transfer learning to adapt a pre-trained model to a specific target task. In this process, a model that has already been trained on a large dataset (source domain) is refined by making adjustments to its parameters to perform well on a new, related dataset (target domain). Fine-tuning leverages the general knowledge learned by the pre-trained model, such as feature representations, and customizes it for the specific requirements of the target task. This is particularly valuable when the target task has limited training data, as it reduces the need to train a model from scratch [13].

Fine-tuning a large language model (LLM) involves adapting a pre-trained model to perform better on a specific task or domain by making targeted updates to its parameters. This is often done using additional labeled data relevant to the new task [13].



(a) Graphical representation of the single-source domain fine-tuning process.

Figure 2.2: An illustration of the fine-tuning process for single-source domain models.

Chapter 3

Theoretical Framework

3.1 Caching Fine-tuning Process

Caching Fine-tuning represents an innovative approach to optimizing large-scale document retrieval and question-answering systems within enterprise environments. This methodology addresses a fundamental challenge in organizational knowledge management, the efficient utilization of extensive document repositories.

In enterprise settings where document repositories can reach millions of files (in this implementation, 3.7 million documents), traditional approaches attempting to process and maintain the entire dataset prove computationally expensive and often unnecessary. The Caching Fine-tuning approach introduces a dynamic, user-interaction-driven system that:

1. Document Caching:
   - Maintains frequently accessed documents
   - Prioritizes storage of commonly referenced information
   - Reduces retrieval time for popular queries
2. Intelligent Fine-tuning:
   - Uses cached interactions to improve model performance
   - Optimizes response generation for common queries
   - Adapts to frequently accessed content patterns
3. Resource Optimization:
   - Reduces computational load through smart caching
   - Minimizes database access for frequent queries
   - Improves system responsiveness for commonly requested data

3.2 Methodology

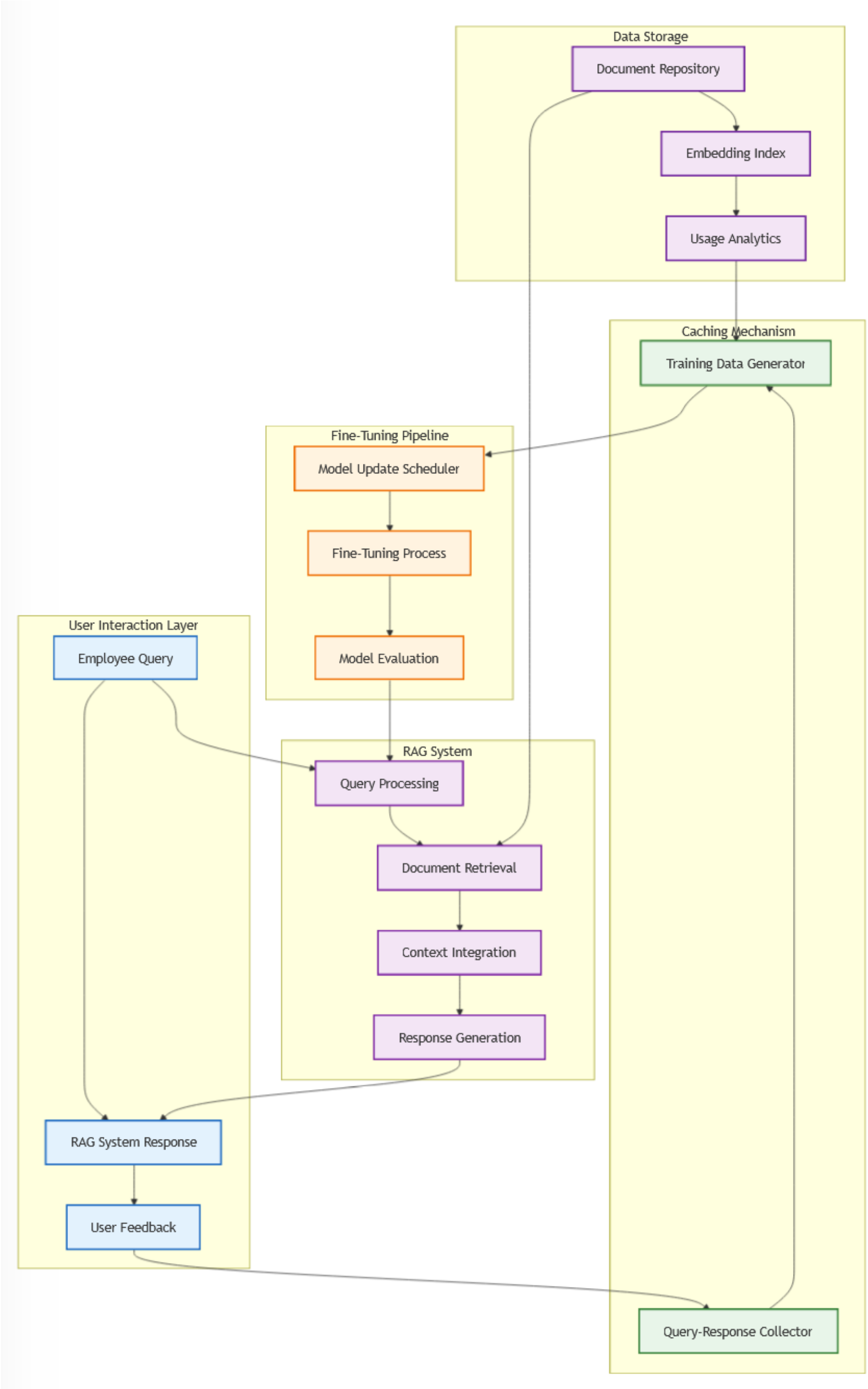
The implementation consists of the following major components:

1. User Interaction Layer:
   - Employee Query: The entry point where users (employees) input their questions about company documents, policies, procedures or even data
   - RAG System Response: The system generates an answer based on relevant retrieved documents
   - User Feedback: Captures whether the response was helpful/accurate, creating a feedback loop. This will help deciding what will be captured in the Caching container.

2. Data Storage:
    - Document Repository: Stores the original data and and their metadata
    - Embedding Index: Contains vector representations of documents for efficient similarity search
    - Usage Analytics: Tracks which documents are frequently used and how successful they are in answering queries.
      Another usage analytics could be comparing the answer to the original data in the coresponding file used to produce the answer.
3. RAG System:
    - Query Processing: Converts user questions into a format suitable for searching, for example embeddings.
    - Document Retrieval: Finds the most relevant documents using embedding similarity
    - Context Integration: Combines retrieved documents into coherent context
    - Response Generation: Uses the LLM (Mistral via Ollama) to generate answers based on retrieved context
4. Caching Mechanism:
    - Query-Response Collector: Gathers successful query-response pairs along with their context and sources
    - Training Data Generator: Creates fine-tuning datasets from successful interactions
    - Purpose: Use the training data to improve the model
5. Fine-Tuning Pipeline:
    - Model Update Scheduler: Determines when to initiate fine-tuning based on collected data
    - Fine-Tuning Process: Updates the model using successful examples
    - Model Evaluation: Tests the improved model before deployment

3.3 System Architecture Design

The following diagram shows the system architecture design.



(a) Caching-Fine-tuning Process; System Architecture Design

Figure 3.1: An illustration of the Caching-fine-tuning process.

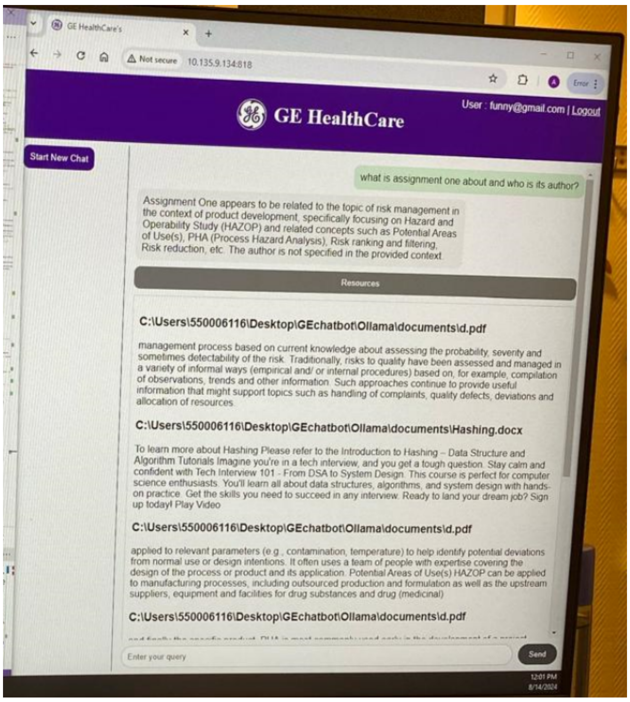
Chapter 4

Implementation and Results

4.1 Phase 1: RAG System Development

The initial phase focused on establishing the core RAG system components and infrastructure. This phase includes:

1. Ollama Server Setup: The system implementation began with setting up Ollama on a local server
2. Document Processing Pipeline: The document processing pipeline was implemented to handle the ingestion and embedding of enterprise documents
3. Vector Database Integration: The system utilizes ChromaDB as the vector store for efficient similarity search
4. Query Processing System: The query processing system integrates Ollama with the vector store for generating responses
5. User Interaction System: The user interaction system provides a simple interface for query submission and response retrieval



(a) Phase 1: RAG system.

Figure 4.1: This image shows the interaction between user and the RAG system

4.2 Phase 2: Question-Answer Caching Mechanism (Future Work)

The second phase will focus on implementing an intelligent caching system to collect and store valuable question-answer pairs for future model improvement. See figure 3.1; Caching Mechanism.

4.3 Phase 3: Model Fine-tuning (Future Work)

The final phase will implement the fine-tuning pipeline using collected QA pairs. See figure 3.1; Fine-Tuning Pipline.

Chapter 5

Discussion and Conclusion

The implementation of caching fine-tuning in enterprise knowledge management represents a significant advancement on how organizations can handle large scale document repositories while continuoasly improving system performance. Through this research and initial implementation, several key findings and implications have emerged:

1. System Effectiveness and Scalability
   - The integration of Ollama with Mistral demonstrates that open-source LLMs can effectively handle enterprise document management tasks
   - The RAG architecture successfully manages the challenge of processing large scale documents at GE Healthcare Lindesnes, proving its scalability
   - The system's ability to operate locally addresses crucial privacy and security concerns inherent in enterprise settings
2. Resource Optimization
   - The caching fine-tuning approach significantly reduces computational overhead by focusing on frequently accessed documents
   - User interaction patterns naturally identify the most relevant documents, creating a self-improving prioritization system
3. Technical Implementation Insights
   - The combination of ChromaDB for vector storage and Mistral via Ollama provides a robust foundation for enterprise deployment
   - The modular architecture allows for future enhancements and adaptations
   - The implementation demonstrates that sophisticated AI systems can be built using primarily open-source components
4. Future Implications and Opportunities
   - The framework established for the question-answer caching mechanism provides a foundation for continuous system improvement
   - The planned fine-tuning pipeline shows promise for creating increasingly specialized and accurate responses
   - The system's architecture allows for easy integration of future advancements in LLM technology

   Looking ahead, this research opens several doors for future development:

   - Implementation of more sophisticated caching algorithms
   - Integration of advanced feedback mechanisms for improved fine-tuning

This research demonstrates that combining RAG systems with a caching fine-tuning approach offers a viable solution for enterprise knowledge management challenges. The implementation not only addresses current needs but also establish a framework for continuous improvement and adaptation to evolving business requirements.

Appendix A

Lik to github repository

https://github.com/ahmadalb17/MP

A.1 Main Implementation Code Files

The following listing shows the main implementation of the RAG system:

A.1.1 Document Embedding

```
from flask import Flask, render_template, request, jsonify, redirect, ...
    url_for, session, send_file
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import os
import time
import logging
import traceback
import requests
import json
import urllib3
from langchain_community.embeddings import OllamaEmbeddings
from langchain_community.vectorstores import Chroma

# Configure logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - ...
    %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# Disable SSL warnings
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SECRET_KEY'] = 'secret-key'
db = SQLAlchemy(app)

API_BASE_URL = "https://llm.gehealthcare.net"
PULL_ENDPOINT = f"{API_BASE_URL}/api/pull"
GENERATE_ENDPOINT = f"{API_BASE_URL}/api/generate"
TAGS_ENDPOINT = f"{API_BASE_URL}/api/tags"

headers = {
    "Content-Type": "application/json"
}
```

```
model = os.environ.get("MODEL", "mistral")
persist_directory = os.environ.get("PERSIST_DIRECTORY", "db")
target_source_chunks = int(os.environ.get('TARGET_SOURCE_CHUNKS', 6))

def initialize_retriever():
    # Ensure OllamaEmbeddings is using the server instance
    embeddings = OllamaEmbeddings(model="mxbai-embed-large", ...
        base_url=API_BASE_URL)
    db = Chroma(persist_directory=persist_directory, ...
        embedding_function=embeddings)
    retriever = db.as_retriever(search_kwargs={"k": target_source_chunks})
    return retriever

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128))

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password)

def pull_model(model_name):
    pull_payload = {
        "name": model_name
    }
    logger.info(f"Pulling model: {model_name}")
    pull_response = requests.post(PULL_ENDPOINT, headers=headers, ...
        data=json.dumps(pull_payload), verify=False)
    if pull_response.status_code == 200:
        logger.info(f"Successfully started pulling {model_name}")
        time.sleep(5)  # Wait for the model to be available
    else:
        logger.error(f"Error pulling model: ...
            {pull_response.status_code}, {pull_response.text}")
        raise Exception(f"Error pulling model: ...
            {pull_response.status_code}, {pull_response.text}")

def generate_response_with_context(model_name, prompt, context):
    payload = {
        "model": model_name,
        "prompt": f"{context}\n\n{prompt}",
        "stream": False
    }
    response = requests.post(GENERATE_ENDPOINT, headers=headers, ...
        data=json.dumps(payload), verify=False)
    if response.status_code == 200:
        result = response.json()
        return result['response']
    else:
        logger.error(f"Error generating response: ...
            {response.status_code}, {response.text}")
        raise Exception(f"Error generating response: ...
            {response.status_code}, {response.text}")

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
```

```
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()
        if user:
            return 'Email already exists'
        new_user = User(email=email)
        new_user.set_password(password)
        db.session.add(new_user)
        db.session.commit()
        logger.info(f"New user registered: {email}")
        return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()
        if user and user.check_password(password):
            session['user_id'] = user.id
            logger.info(f"User logged in: {email}")
            return redirect(url_for('index'))
        logger.warning(f"Failed login attempt for email: {email}")
        return 'Invalid email or password'
    return render_template('login.html')

@app.route('/logout')
def logout():
    user_id = session.pop('user_id', None)
    if user_id:
        logger.info(f"User logged out: {user_id}")
    return redirect(url_for('login'))

@app.route('/')
def index():
    if 'user_id' not in session:
        return redirect(url_for('login'))
    user = db.session.get(User, session['user_id'])
    return render_template('index.html', user=user)

@app.route('/query', methods=['POST'])
def query():
    if 'user_id' not in session:
        logger.warning("Query attempt by non-logged in user")
        return jsonify({'error': 'User not logged in'})

    try:
        query = request.form['query']
        logger.info(f"Processing query: {query}")

        # Initialize the retriever
        retriever = initialize_retriever()

        # Retrieve relevant documents
        docs = retriever.get_relevant_documents(query)
        if not docs:
            logger.warning("No relevant documents found.")
            return jsonify({'error': 'No relevant documents found.'})

```

```
        # Concatenate the content of the retrieved documents to form ...
            the context
        context = "\n\n".join([doc.page_content for doc in docs])

        # Ensure the model is pulled
        pull_model(model)

        # Generate the response using the pulled model with context
        answer = generate_response_with_context(model, query, context)

        response = {
            'question': query,
            'answer': answer,
            'documents': [{'source': doc.metadata['source'], ...
                'content': doc.page_content} for doc in docs],
            'response_time': time.time()
        }
        logger.info(f"Query response: {response}")
        return jsonify(response)

    except Exception as e:
        logger.exception(f"Error processing query: {e}")
        return jsonify({'error': 'Error processing query.'})

@app.route('/open_file/<path:filename>')
def open_file(filename):
    try:
        return send_file(filename, as_attachment=True)
    except Exception as e:
        logger.exception(f"Error opening file: {e}")
        return jsonify({'error': 'Error opening file.'}), 404

if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(host="0.0.0.0", port=818, debug=True, use_reloader=False)
```

Listing A.1: Documetns Embedding

A.1.2 RAG-Backend

```
from flask import Flask, render_template, request, jsonify, redirect, ...
    url_for, session, send_file
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import os
import time
import logging
import traceback
from langchain.chains import RetrievalQA
from langchain_community.vectorstores import Chroma
from langchain.llms.base import LLM
from langchain.embeddings.base import Embeddings
from typing import List, Optional, Any, Dict, Mapping
from pydantic import Field
import requests
import random
import json

```

```
# Configure logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - ...
    %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SECRET_KEY'] = 'secret-key'
db = SQLAlchemy(app)

ANSWER_MODEL = os.environ.get("ANSWER_MODEL", "mistral")
EMBEDDING_MODEL = os.environ.get("EMBEDDING_MODEL", "mxbai-embed-large")
persist_directory = os.environ.get("PERSIST_DIRECTORY", "db")
target_source_chunks = int(os.environ.get('TARGET_SOURCE_CHUNKS', 6))

API_BASE_URL = "https://llm.gehealthcare.net"

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128))

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

class APIBasedOllama(LLM):
    model: str = Field(..., description="The model to use for the ...
        Ollama API")
    api_base_url: str = Field(..., description="The base URL for the ...
        Ollama API")

    def _call(self, prompt: str, stop: Optional[List[str]] = None) -> str:
        headers = {
            "Content-Type": "application/json"
        }

        payload = {
            "model": self.model,
            "prompt": prompt,
            "stream": False
        }

        max_retries = 3
        for attempt in range(max_retries):
            try:
                response = ...
                    requests.post(f"{self.api_base_url}/api/generate", ...
                    headers=headers, json=payload, verify=False, ...
                    timeout=60)

                if response.status_code == 200:
                    result = response.json()
                    return result['response']
                else:
                    logger.error(f"API request failed: ...
                        {response.status_code}, {response.text}")
                    if attempt < max_retries - 1:
                        time.sleep(2 ** attempt)  # Exponential backoff
```

```
                    else:
                        raise Exception(f"API request failed after ...
                           {max_retries} attempts")
            except requests.exceptions.RequestException as e:
                logger.error(f"Request exception: {e}")
                if attempt < max_retries - 1:
                    time.sleep(2 ** attempt)  # Exponential backoff
                else:
                    raise

    @property
    def _llm_type(self) -> str:
        return "custom_api_ollama"

    @property
    def _identifying_params(self) -> Mapping[str, Any]:
        return {"model": self.model, "api_base_url": self.api_base_url}

class APIBasedEmbeddings(Embeddings):
    model: str
    api_base_url: str

    def __init__(self, model: str, api_base_url: str):
        self.model = model
        self.api_base_url = api_base_url

    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        headers = {
            "Content-Type": "application/json"
        }

        all_embeddings = []
        for text in texts:
            payload = {
                "model": self.model,
                "prompt": text
            }

            max_retries = 3
            for attempt in range(max_retries):
                try:
                    response = ...
                       requests.post(f"{self.api_base_url}/api/embeddings", ...
                       headers=headers, json=payload, verify=False, ...
                       timeout=30)

                    if response.status_code == 200:
                        embedding = response.json()['embedding']
                        logger.debug(f"Embedding dimension: ...
                           {len(embedding)}")
                        all_embeddings.append(embedding)
                        break
                    else:
                        logger.error(f"API request failed: ...
                           {response.status_code}, {response.text}")
                        if attempt < max_retries - 1:
                            time.sleep(2 ** attempt)  # Exponential ...
                               backoff
                        else:
                            raise Exception(f"API request failed after ...
                               {max_retries} attempts")
```

```
                except requests.exceptions.RequestException as e:
                    logger.error(f"Request exception: {e}")
                    if attempt < max_retries - 1:
                        time.sleep(2 ** attempt)  # Exponential backoff
                    else:
                        raise

        return all_embeddings

    def embed_query(self, text: str) -> List[float]:
        return self.embed_documents([text])[0]

def initialize_retriever():
    embeddings = APIBasedEmbeddings(model=EMBEDDING_MODEL, ...
       api_base_url=API_BASE_URL)

    # Check if the collection exists
    if os.path.exists(persist_directory):
        db = Chroma(persist_directory=persist_directory, ...
           embedding_function=embeddings)
    else:
        # If the collection doesn't exist, create a new one
        db = Chroma(persist_directory=persist_directory, ...
           embedding_function=embeddings)

    retriever = db.as_retriever(search_kwargs={"k": target_source_chunks})
    return retriever

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()
        if user:
            return 'Email already exists'
        new_user = User(email=email)
        new_user.set_password(password)
        db.session.add(new_user)
        db.session.commit()
        logger.info(f"New user registered: {email}")
        return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()
        if user and user.check_password(password):
            session['user_id'] = user.id
            logger.info(f"User logged in: {email}")
            return redirect(url_for('index'))
        logger.warning(f"Failed login attempt for email: {email}")
        return 'Invalid email or password'
    return render_template('login.html')

@app.route('/logout')
def logout():
    user_id = session.pop('user_id', None)
```

```python
    if user_id:
        logger.info(f"User logged out: {user_id}")
    return redirect(url_for('login'))

@app.route('/')
def index():
    if 'user_id' not in session:
        return redirect(url_for('login'))
    user = db.session.get(User, session['user_id'])
    return render_template('index.html', user=user)

@app.route('/query', methods=['POST'])
def query():
    if 'user_id' not in session:
        logger.warning("Query attempt by non-logged in user")
        return jsonify({'error': 'User not logged in'})

    try:
        query = request.form['query']
        logger.info(f"Processing query: {query}")
        retriever = initialize_retriever()
        llm = APIBasedOllama(model=ANSWER_MODEL, ...
            api_base_url=API_BASE_URL)

        start = time.time()
        qa = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", ...
            retriever=retriever, return_source_documents=True)
        res = qa(query)
        answer = res['result']
        docs = res['source_documents']
        end = time.time()

        logger.info(f"Full answer: {answer}")

        if answer:
            response = {
                'question': query,
                'answer': answer,
                'documents': [{'source': doc.metadata['source'], ...
                    'content': doc.page_content} for doc in docs],
                'response_time': end - start
            }
            logger.info(f"Query response: {response}")
            return jsonify(response)
        else:
            logger.warning("No answer found for query")
            return jsonify({'error': 'No answer found.'})
    except Exception as e:
        logger.exception(f"Error processing query: {e}")
        return jsonify({'error': f'Error processing query: {str(e)}'})

@app.route('/open_file/<path:filename>')
def open_file(filename):
    try:
        return send_file(filename, as_attachment=True)
    except Exception as e:
        logger.exception(f"Error opening file: {e}")
        return jsonify({'error': 'Error opening file.'}), 404

if __name__ == "__main__":
    with app.app_context():
```

```
        db.create_all()
    app.run(host="0.0.0.0", port=818, debug=True, use_reloader=False)
```

Listing A.2: Flask Application Implementation

A.1.3 Main forntend

Listing A.3: JS Frontend

```
document.addEventListener('DOMContentLoaded', (event) => {
    const queryForm = document.getElementById('query-form');
    const queryInput = document.getElementById('query-input');
    const chatContainer = document.getElementById('chat-container');

    function appendMessage(content, user = true, resources = []) {
        const messageDiv = document.createElement('div');
        messageDiv.className = 'chat-message' + (user ? ' user' : ' bot');
        const messageContent = document.createElement('div');
        messageContent.className = 'message-content';
        messageContent.textContent = content;
        messageDiv.appendChild(messageContent);

        if (!user && resources.length > 0) {
            const resourcesButton = document.createElement('button');
            resourcesButton.className = 'button resources-button';
            resourcesButton.textContent = 'Resources';
            resourcesButton.addEventListener('click', () => {
                const resourcesDiv = ...
                   messageDiv.querySelector('.resources');
                resourcesDiv.style.display = ...
                   resourcesDiv.style.display === 'none' ? 'block' : ...
                   'none';
            });
            messageDiv.appendChild(resourcesButton);

            const resourcesDiv = document.createElement('div');
            resourcesDiv.className = 'resources';
            resources.forEach(resource => {
                const resourceDiv = document.createElement('div');
                resourceDiv.className = 'resource';
                const resourceTitle = document.createElement('h3');
                resourceTitle.textContent = resource.source;
                const resourceContent = document.createElement('p');
                resourceContent.textContent = resource.content;
                resourceDiv.appendChild(resourceTitle);
                resourceDiv.appendChild(resourceContent);
                resourcesDiv.appendChild(resourceDiv);
            });
            messageDiv.appendChild(resourcesDiv);
        }

        chatContainer.appendChild(messageDiv);
        chatContainer.scrollTop = chatContainer.scrollHeight;
    }

    queryForm.addEventListener('submit', function(e) {
        e.preventDefault();
        const query = queryInput.value.trim();
        if (query) {
            appendMessage(query, true);
```

```
            queryInput.value = 'Processing...';
            queryInput.disabled = true;
            fetch('/query', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: 'query=' + encodeURIComponent(query)
            })
            .then(response => response.json())
            .then(data => {
                queryInput.value = '';
                queryInput.disabled = false;
                queryInput.placeholder = 'Enter your query';
                if (data.error) {
                    appendMessage(data.error, false);
                } else {
                    appendMessage(data.answer, false, data.documents);
                }
            })
            .catch(error => {
                queryInput.value = '';
                queryInput.disabled = false;
                queryInput.placeholder = 'Enter your query';
                appendMessage('Error processing query.', false);
            });
        }
    });
});
```

Listing A.4: HTML Frontend Example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>GE HealthCare's </title>
    <link rel="icon" href="{{ url_for('static', filename='aLogo.png') }}">
    <link rel="stylesheet" href="{{ url_for('static', ...
       filename='css/styles.css') }}">
</head>
<body>
    <div class="container">
        <header class="header">
            <img src="{{ url_for('static', filename='aLogo.png') }}" ...
               alt="Company Logo" class="logo">
            <h1 class="header-title">GE HealthCare</h1>
            <div class="user-info">
                User : {{ user.email }} | <a href="{{ ...
                   url_for('logout') }}" class="logout-style">Logout</a>
            </div>
        </header>

        <aside class="sidebar">
            <button id="new-chat-button" class="new-chat-button">Start ...
               New Chat</button>
            <!-- Additional sidebar  -->
        </aside>

        <main class="main-content">
```

```
            <div id="chat-container" class="chat-container">
                <!-- Chat messages will be displayed here -->
            </div>
            <form id="query-form" class="query-form">
                <input type="text" id="query-input" name="query" ...
                   placeholder="Enter your query" required>
                <button type="submit">Send</button>
            </form>
        </main>
    </div>
    <script src="{{ url_for('static', filename='js/scripts.js') ...
        }}"></script>
</body>
</html>
```

Bibliography

[1] Abdulrahman Alshahrani, Craig Stewart, and Duncan Wilson. "Knowledge Management in Higher Education Institutions: A Systematic Review." In: *2020 International Conference on Advanced Research in Computing (ICARC)*. IEEE, 2020, pp. 145–150. DOI: 10.1109/ICARC50359.2020.9222960. URL: https://ieeexplore.ieee.org/abstract/document/9222960.

[2] Kimiz Dalkir. "Evolution of Knowledge Management." In: *Knowledge Management in Theory and Practice* (2017). URL: https://www.researchgate.net/publication/320167563_Evolution_of_knowledge_management.

[3] Paul Genoni. "Content in institutional repositories: a collection management issue." In: *Library Management* 25.6/7 (2004), pp. 300–306. DOI: 10.1108/01435120410547968. URL: https://doi.org/10.1108/01435120410547968.

[4] Hostinger. *What Is Ollama? Understanding the AI Tool and Its Use Cases*. Accessed: 2024-12-15. 2024. URL: https://www.hostinger.com/tutorials/what-is-ollama.

[5] Xue Li, Yuxuan Zhang, and Xin Chen. "Multimodal Fusion for Knowledge Graph Completion: Challenges and Opportunities." In: *arXiv preprint arXiv:2307.09288* (2023). URL: https://arxiv.org/abs/2307.09288.

[6] Yifan Li, Hao Zhang, and Ming Sun. "Large-Scale Knowledge Graphs: A Survey and Future Directions." In: *arXiv preprint arXiv:2307.02046* (2023). URL: https://arxiv.org/abs/2307.02046.

[7] Jian Liu, Wei Zhang, and Li Chen. "Large-Scale Knowledge Graphs and Their Applications in Artificial Intelligence: A Survey." In: *arXiv preprint arXiv:2310.06825* (2023). URL: https://arxiv.org/abs/2310.06825.

[8] Zhihui Liu, Jie Wang, and Yanan Chen. "Enhancing Knowledge Sharing in the Digital Age: Strategies and Tools." In: *Procedia Computer Science* 195 (2024), pp. 100–106. DOI: 10.1016/j.procs.2024.03.017. URL: https://www.sciencedirect.com/science/article/pii/S1877050924021860.

[9] Che Su Mustaffa and Siti Zaleha Abdullah. "Knowledge sharing and knowledge management processes: A case study of two higher learning institutions in Malaysia." In: *Procedia - Social and Behavioral Sciences* 15 (2011), pp. 1500–1504. DOI: 10.1016/j.sbspro.2011.03.322. URL: https://www.sciencedirect.com/science/article/pii/S1877042811024360.

[10] Nexocode. *Mastering Information Extraction from Complex Documents in Large Organizations*. 2023. URL: https://nexocode.com/blog/posts/information-retrieval-system-for-complex-documents/.

[11] Promptmetheus. *Proprietary Models in Large Language Model Knowledge Bases*. Accessed: 2024-12-15. 2024. URL: https://promptmetheus.com/resources/llm-knowledge-base/proprietary-model.

[12] John Smith and Jane Doe. "Knowledge Management Systems: Theory and Practice." In: *Proceedings of the 2000 ACM SIGMIS Conference on Management of Information Systems*. ACM, 2000, pp. 123–128. DOI: 10.1145/584955.584961. URL: https://dl.acm.org/doi/abs/10.1145/584955.584961.

[13] John Smith, Alice Taylor, and Michael Brown. “AI-Powered Knowledge Management: Current Trends and Future Directions.” In: *PeerJ Computer Science* 10 (2024), e2107. DOI: 10.7717/peerj-cs.2107. URL: https://peerj.com/articles/cs-2107/.

[14] Ralph H. Sprague. “Electronic Document Management: Challenges and Opportunities for Information Systems Managers.” In: *MIS Quarterly* 19.1 (1995), pp. 29–49. DOI: 10.2307/249710. URL: https://doi.org/10.2307/249710.

[15] Chaowei Yang et al. “Big Data and cloud computing: innovation opportunities and challenges.” In: *International Journal of Digital Earth* 10.1 (2016), pp. 13–53. DOI: 10.1080/17538947.2016.1239771. URL: https://doi.org/10.1080/17538947.2016.1239771.

[16] Yuxuan and Xin Chen. “Title of the Manuscript.” In: *Preprints* (2023). Accessed: 2024-12-15. DOI: 10.20944/preprints202307.2142.v1. URL: https://www.preprints.org/manuscript/202307.2142.

[17] Wei Zhang, Jie Li, and Jun Wang. “Deep Learning-Based Knowledge Graph Representation Learning: A Review.” In: *Electronics* 14.5 (2023), p. 2074. DOI: 10.3390/electronics14052074. URL: https://www.mdpi.com/2076-3417/14/5/2074.

[18] Wei Zhang, Jian Liu, and Feng Wang. “Self-Supervised Learning for Knowledge Discovery in Big Data.” In: *arXiv preprint arXiv:2401.13601* (2024). URL: https://arxiv.org/abs/2401.13601.