

# Laboratory Handbook: Building a Local RAG System with Flask and MLflow

RYAN HAMMANG

Creating a local RAG system with Flask and MLflow.

## ACM Reference Format:

Ryan Hammang. 2025. Laboratory Handbook: Building a Local RAG System with Flask and MLflow. 1, 1 (March 2025), 106 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

## Project Overview

This laboratory course guides you through building a complete Retrieval-Augmented Generation (RAG) system that operates entirely on local hardware. By the end of this intensive workshop, you will have created a production-ready application that processes PDF documents, generates embeddings, performs semantic search, and answers questions using local large language models all accessible through a Flask web interface and MLflow serving endpoint.

## Hardware Requirements

- Apple MacMini with M2 Pro chip (or equivalent)
- 16GB RAM minimum
- 50GB available storage
- macOS Ventura 15.3.1 or later

## Software Prerequisites

- Python 3.10+
- Docker Desktop for Mac
- Git
- Homebrew (recommended for macOS)

## Lab 1: Foundation and Infrastructure

### Lab 1.1: Environment Setup and Project Structure.

---

Author's address: Ryan Hammang.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM XXXX-XXXX/2025/3-ART

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

*Learning Objectives:*

- Configure a proper development environment for ML applications
- Understand containerization principles with Docker
- Establish a maintainable project structure

*Setup Instructions:*

1. Open Terminal and create your project directory structure:

```
mkdir -p {app,data,models,docker,mlflow,flask-app}
mkdir -p data/{pdfs,vectors,documents}
mkdir -p models/{embedding,reranker,llm}
mkdir -p app/{utils,api,evaluation,tests}
mkdir -p flask-app/{static,templates,utils,tests}
mkdir -p flask-app/static/{css,js,images}
```

2. Create and activate a virtual environment:

```
python -m venv venv
source venv/bin/activate
pip install --upgrade pip wheel setuptools
```

3. Create a requirements file:

```
touch requirements.txt
```

4. Add the following dependencies to `requirements.txt`:

```
# Core Libraries
pandas==2.2.1
numpy==1.26.4
scikit-learn==1.6.1 # Updated to latest available for Python 3.12 on macOS ARM64

# PDF Processing
pymupdf==1.24.5
pdfminer.six==20231228
tqdm==4.66.4

# Text Processing
langchain==0.2.7
langchain-community==0.2.7
llama-index==0.10.32

# Embedding Models - updated for Python 3.12
sentence-transformers==2.6.0
transformers==4.39.3
torch==2.6.0

# Vector Database
faiss-cpu==1.8.0
qdrant-client==1.8.0
```

```

95 chromadb==0.4.22
96
97 # LLM Inference - optimized for Apple Silicon
98 llama-cpp-python==0.2.49
99
100 # MLflow
101 mlflow==2.12.2
102 protobuf==4.25.3
103 alembic==1.13.1
104 sqlalchemy==2.0.28
105
106 # Flask Web UI
107 flask==2.3.3
108 flask-wtf==1.2.1
109 werkzeug==2.3.7
110 jinja2==3.1.3
111 itsdangerous==2.1.2
112
113 # API and Testing
114 requests==2.31.0
115 pytest==7.4.4
116 locust==2.20.1
117
118 # Utilities
119 python-dotenv==1.0.1
120 click==8.1.7

```

5. Install the dependencies:

```

121 pip install --upgrade pip wheel setuptools
122 pip uninstall -y llama-cpp-python
123 CMAKE_ARGS="-DLLAMA_METAL=on" pip install llama-cpp-python==0.2.49
124 pip install --prefer-binary -r requirements.txt
125

```

**Professor's Hint:** When working with ML libraries on Apple Silicon, use native ARM packages where possible. The torch package specified here is compiled for M-series chips. For libraries without native ARM support, Rosetta 2 will handle the translation, but with a performance penalty.

6. Create a Docker Compose file in the root directory:

```

131 touch docker-compose.yml
132

```

7. Add the following configuration to docker-compose.yml:

```

133 # docker-compose.yml - Updated to avoid port 5000 conflict
134
135 services:
136   vector-db:
137     image: qdrant/qdrant:latest
138     ports:

```

```

142     - "6333:6333"
143     - "6334:6334"
144 volumes:
145     - ./data/vectors:/qdrant/storage
146 environment:
147     - QDRANT_ALLOW_CORS=true
148 restart: unless-stopped
149 healthcheck:
150     test: ["CMD", "curl", "-f", "http://localhost:6333/health"]
151     interval: 30s
152     timeout: 10s
153     retries: 3
154     start_period: 10s
155 mlflow:
156     image: ghcr.io/mlflow/mlflow:latest
157     ports:
158     - "5001:5000" # External port 5001 mapped to container's 5000
159 volumes:
160     - ./mlflow/artifacts:/mlflow/artifacts
161     - ./mlflow/backend:/mlflow/backend
162 environment:
163     - MLFLOW_TRACKING_URI=sqlite:///mlflow/backend/mlflow.db
164     - MLFLOW_DEFAULT_ARTIFACT_ROOT=/mlflow/artifacts
165 command: mlflow server \
166     --backend-store-uri sqlite:///mlflow/backend/mlflow.db \
167     --default-artifact-root /mlflow/artifacts \
168     --host 0.0.0.0 \
169     --port 5000
170 restart: unless-stopped
171 healthcheck:
172     test: ["CMD", "curl", "-f", "http://localhost:5000/ping"]
173     interval: 30s
174     timeout: 10s
175     retries: 3
176     start_period: 10s
177 flask-app:
178     build:
179     context: ./flask-app
180     dockerfile: Dockerfile
181 ports:
182     - "8000:8000"
183 volumes:
184     - ./flask-app:/app
185     - ./data:/data
186     - ./app:/rag_app
187 environment:
188     - FLASK_APP=app.py
189     - FLASK_DEBUG=1

```

```

189     - PYTHONPATH=/app:/rag_app
190     - MLFLOW_HOST=mlflow # Use service name for Docker networking
191     - MLFLOW_PORT=5001   # External port for client connections
192 depends_on:
193     vector-db:
194         condition: service_healthy
195     mlflow:
196         condition: service_healthy
197 restart: unless-stopped
198 volumes:
199     mlflow-data:
200         driver: local
201     vector-data:
202         driver: local
203 networks:
204     default:
205         driver: bridge
206         name: rag-network
207

```

8. Start the containers to verify the setup:

```
docker-compose up -d
```

#### Checkpoint Questions:

1. Why are we using Docker for certain components instead of running everything natively?
2. What is the purpose of volume mapping in Docker Compose?
3. What does the `restart: unless-stopped` directive do in the Docker Compose file?
4. How does containerization affect the portability versus performance tradeoff for ML systems?
5. What security implications arise from running AI models locally versus in cloud environments?
6. How would different embedding model sizes affect the system's memory footprint and performance?
7. **Additional Challenge:** Add a PostgreSQL container to the Docker Compose file as an alternative backend for MLflow instead of SQLite.

#### Lab 1.2: Model Preparation and Configuration.

##### Learning Objectives:

- Download and prepare ML models for offline use
- Create configuration files for application settings
- Understand model size and performance tradeoffs

##### Exercises:

1. Create a configuration file for the application:

```

mkdir -p app/config
touch app/config/settings.py

```

2. Add the following to `app/config/settings.py`:

```
import os
from pathlib import Path

# Base directory
BASE_DIR = Path(__file__).resolve().parent.parent

# Vector database settings
VECTOR_DB_HOST = "localhost"
VECTOR_DB_PORT = 6333
VECTOR_DIMENSION = 384 # For all-MiniLM-L6-v2
COLLECTION_NAME = "pdf_chunks"

# Document processing
CHUNK_SIZE = 500
CHUNK_OVERLAP = 50
MAX_CHUNKS_PER_DOC = 1000

# Model paths
EMBEDDING_MODEL_PATH = os.path.join(BASE_DIR, "models", "embedding", "all-MiniLM-L6-v2")
RERANKER_MODEL_PATH = os.path.join(BASE_DIR, "models", "reranker", "ms-marco-MiniLM-L-6-v2")
LLM_MODEL_PATH = os.path.join(BASE_DIR, "models", "llm", "llama-2-7b-chat-q4_0.gguf")

# MLflow settings
MLFLOW_TRACKING_URI = "http://localhost:5001"
MLFLOW_MODEL_NAME = "rag_model"

# Flask settings
FLASK_SECRET_KEY = "change-this-in-production"
PDF_UPLOAD_FOLDER = os.path.join(BASE_DIR, "data", "pdfs")
ALLOWED_EXTENSIONS = {'pdf'}
```

3. Create a script to download the embedding model:

```
mkdir -p app/scripts
touch app/scripts/download_models.py
```

4. Add the following to `app/scripts/download_models.py`:

```
import os
import sys
from pathlib import Path

# Add the project root to the Python path
sys.path.insert(0, str(Path(__file__).resolve().parent.parent))

from app.config.settings import EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH
import torch

def download_embedding_model():
```

```

283     """Download the embedding model for offline use."""
284     try:
285         from sentence_transformers import SentenceTransformer
286
287         print(f"Downloading embedding model to {EMBEDDING_MODEL_PATH}...")
288         model = SentenceTransformer('all-MiniLM-L6-v2')
289         os.makedirs(os.path.dirname(EMBEDDING_MODEL_PATH), exist_ok=True)
290         model.save(EMBEDDING_MODEL_PATH)
291         print("Embedding model downloaded successfully.")
292
293         # Test the model
294         test_embedding = model.encode(["Hello world"])
295         print(f"Test embedding shape: {test_embedding.shape}")
296
297     except Exception as e:
298         print(f"Error downloading embedding model: {str(e)}")
299         sys.exit(1)
300
301 def download_reranker_model():
302     """Download the reranker model for offline use."""
303     try:
304         from sentence_transformers import CrossEncoder
305
306         print(f"Downloading reranker model to {RERANKER_MODEL_PATH}...")
307         model = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')
308         os.makedirs(os.path.dirname(RERANKER_MODEL_PATH), exist_ok=True)
309         model.save(RERANKER_MODEL_PATH)
310         print("Reranker model downloaded successfully.")
311
312     except Exception as e:
313         print(f"Error downloading reranker model: {str(e)}")
314         sys.exit(1)
315
316 if __name__ == "__main__":
317     download_embedding_model()
318     download_reranker_model()
319     print("Please manually download the LLM model using the instructions in the README.")

```

5. Run the script to download the models:

```
python app/scripts/download_models.py
```

6. Create a script to download the LLM model (manual step due to size):

```
touch app/scripts/download_llm.sh
chmod +x app/scripts/download_llm.sh
```

7. Add the following to app/scripts/download\_llm.sh:

```
#!/bin/bash
```

```

330
331 # Directory for LLM model
332 LLM_DIR="models/llm"
333 mkdir -p $LLM_DIR
334
335 # URL for Llama-2-7B-Chat quantized model
336 #!/bin/bash
337
338 # Directory for LLM model
339 LLM_DIR="models/llm"
340 mkdir -p $LLM_DIR
341
342 # URL for Llama-3-8B-Instruct quantized model (latest as of now)
343 MODEL_URL="https://huggingface.co/TheBloke/Llama-3-8B-Instruct-GGUF/resolve/main/llama-3-8b-instruct.Q4_K_M.gguf"
344 OUTPUT_PATH="$LLM_DIR/llama-3-8b-instruct-q4.gguf"
345
346 echo "Downloading Llama 3 model to $OUTPUT_PATH..."
347 echo "This may take some time depending on your internet connection."
348
349 # Download with curl, showing progress
350 curl -L -o "$OUTPUT_PATH" "$MODEL_URL" --progress-bar
351
352 echo "Download complete. Verifying file..."
353
354 # Check if file exists and has content
355 if [ -f "$OUTPUT_PATH" ] && [ -s "$OUTPUT_PATH" ]; then
356     echo "Llama 3 model downloaded successfully."
357 else
358     echo "Error: Llama 3 model download failed or file is empty."
359     exit 1
360 fi

```

## 7.5 Alternative using Meta's Llama-3.2-3B-Instruct model:

```

361 ./app/scripts/download_llm.sh
362 #!/bin/bash
363
364 # Directory for LLM model
365 LLM_DIR="models/llm"
366 mkdir -p $LLM_DIR
367
368 # Prompt for Hugging Face token
369 if [ -z "$HF_TOKEN" ]; then
370     echo "Please enter your Hugging Face token (from https://huggingface.co/settings/tokens):"
371     read -s HF_TOKEN
372     echo
373 fi
374
375 # Download using huggingface-cli
376 echo "Installing huggingface_hub if needed..."

```



```

377 pip install -q huggingface_hub
378
379 echo "Downloading Llama-3.2-3B-Instruct model..."
380 echo "This will take some time depending on your connection."
381
382 python -c "
383 from huggingface_hub import snapshot_download
384 import os
385
386 # Set token
387 os.environ['HF_TOKEN'] = '$HF_TOKEN'
388
389 # Download model files
390 model_path = snapshot_download(
391     repo_id='meta-llama/Llama-3.2-3B-Instruct',
392     local_dir='$LLM_DIR/Llama-3.2-3B-Instruct',
393     local_dir_use_symlinks=False
394 )
395
396 print(f'Model downloaded to {model_path}')
397 "
398
399 # Update settings.py to use this model
400
401 SETTINGS_PATH="app/config/settings.py"
402 if [ -f "$SETTINGS_PATH" ]; then
403     if grep -q "LLM_MODEL_PATH" "$SETTINGS_PATH"; then
404         sed -i 's|LLM_MODEL_PATH = .*|LLM_MODEL_PATH = os.path.join(BASE_DIR, "models",
405             "llm", "Llama-3.2-3B-Instruct")|' "$SETTINGS_PATH"
406         echo "Updated settings.py to use the Llama-3.2-3B-Instruct model."
407     fi
408 fi
409
410 echo "Now installing transformers to use with Meta's model format..."
411 pip install -q transformers accelerate
412
413 # Create a model loader adapter
414 mkdir -p app/utils/adapters
415 cat > app/utils/adapters/meta_llama_adapter.py << 'EOF'
416 import os
417 import logging
418 from typing import Dict, Any, List
419 from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
420
421 logger = logging.getLogger(__name__)
422
423 class MetaLlamaAdapter:
424     def __init__(self, model_path: str, max_new_tokens: int = 512):
425         """Initialize the Meta Llama adapter."""
426         logger.info(f>Loading Meta Llama model from {model_path}")

```

```

424     self.model_path = model_path
425     self.max_new_tokens = max_new_tokens
426
427     # Load model and tokenizer
428     self.tokenizer = AutoTokenizer.from_pretrained(model_path)
429     self.model = AutoModelForCausalLM.from_pretrained(
430         model_path,
431         device_map="auto",
432         torch_dtype="auto",
433         low_cpu_mem_usage=True
434     )
435
436     # Create pipeline
437     self.pipe = pipeline(
438         "text-generation",
439         model=self.model,
440         tokenizer=self.tokenizer
441     )
442
443     logger.info("Meta Llama model loaded successfully")
444
445 def __call__(self, prompt: str, **kwargs):
446     """Generate text using the Meta Llama model."""
447     generation_kwargs = {
448         "max_new_tokens": kwargs.get("max_tokens", self.max_new_tokens),
449         "temperature": kwargs.get("temperature", 0.2),
450         "top_p": kwargs.get("top_p", 0.9),
451         "do_sample": kwargs.get("temperature", 0.2) > 0,
452     }
453
454     # Generate text
455     outputs = self.pipe(
456         prompt,
457         **generation_kwargs
458     )
459
460     # Format to match llama-cpp-python output format
461     generated_text = outputs[0]["generated_text"][len(prompt):]
462
463     # Count tokens
464     input_tokens = len(self.tokenizer.encode(prompt))
465     output_tokens = len(self.tokenizer.encode(generated_text))
466
467     return {
468         "choices": [
469             {
470                 "text": generated_text,
471                 "finish_reason": "length" if output_tokens >=
generation_kwargs["max_new_tokens"] else "stop",

```

```

471         ],
472         "usage": {
473             "prompt_tokens": input_tokens,
474             "completion_tokens": output_tokens,
475             "total_tokens": input_tokens + output_tokens,
476         },
477     }
478 EOF
479 # Update llm.py to use the Meta Llama adapter
480 sed -i 's/from llama_cpp import Llama/from llama_cpp import Llama\nfrom
481     app.utils.adapters.meta_llama_adapter import MetaLlamaAdapter/' app/utils/llm.py
482
483 # Update the LLMProcessor.__init__ method
484 sed -i 's/self.model = Llama(/# Check if using Meta Llama model\n        if "Llama-3" in
485     model_path and os.path.isdir(model_path):\n            self.model = MetaLlamaAdapter(\n
486         model_path=model_path,\n                max_new_tokens=max_tokens\n
487     )\n        else:\n            # Use llama-cpp for GGUF models\n            self.model =
488     Llama(/' app/utils/llm.py
489
490 echo "Setup complete for using meta-llama/Llama-3.2-3B-Instruct"

```

8. Run the script to download the LLM model:

```
./app/scripts/download_llm.sh
```

**Professor's Hint:** The LLM download is about 4GB, so it may take some time. We're using a 4-bit quantized model (Q4\_0) to optimize for memory usage on the MacMini. The quality-performance tradeoff is reasonable for most use cases. For higher quality, consider the Q5\_K\_M variant if your system has sufficient RAM.

*Checkpoint Questions:*

1. Why do we use a configuration file instead of hardcoding values in our application?
2. What is model quantization and why is it important for local LLM deployment?
3. How does the dimension of the embedding vector (384) impact our system?
4. **Additional Challenge:** Create a script to benchmark the performance of the embedding model and LLM on your local hardware. Measure throughput (tokens/second) and memory usage.

## Lab 2: PDF Processing and Embedding Pipeline

### Lab 2.1: PDF Ingestion and Text Extraction.

*Learning Objectives:*

- Implement robust PDF text extraction
- Handle various PDF formats and structures
- Build a scalable document processing pipeline

*Exercises:*

1. Create a PDF ingestion utility:

```
touch app/utils/pdf_ingestion.py
```

2. Add the following to app/utils/pdf\_ingestion.py:

```
import os
import pandas as pd
from pathlib import Path
from typing import List, Dict, Any
import fitz # PyMuPDF
from tqdm import tqdm
import logging

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

def scan_directory(directory_path: str) -> List[Dict[str, Any]]:
    """
    Scan a directory for PDF files.

    Args:
        directory_path: Path to the directory containing PDF files

    Returns:
        List of dictionaries with PDF file information
    """
    logger.info(f"Scanning directory: {directory_path}")
    pdf_files = []

    for path in tqdm(list(Path(directory_path).rglob('*.pdf'))):
        try:
            # Get basic file information
            file_info = {
                'path': str(path),
                'filename': path.name,
                'parent_dir': str(path.parent),
                'size_bytes': path.stat().st_size,
                'last_modified': path.stat().st_mtime
            }

            # Get PDF-specific metadata if possible
            try:
                with fitz.open(str(path)) as doc:
                    file_info['page_count'] = len(doc)
                    file_info['metadata'] = doc.metadata
            except Exception as e:
```

```

565         logger.warning(f"Could not read PDF metadata for {path}: {str(e)}")
566         file_info['page_count'] = 0
567         file_info['metadata'] = {}
568
569         pdf_files.append(file_info)
570     except Exception as e:
571         logger.error(f"Error processing {path}: {str(e)}")
572
573     logger.info(f"Found {len(pdf_files)} PDF files")
574     return pdf_files
575
576 def create_pdf_dataframe(pdf_files: List[Dict[str, Any]]) -> pd.DataFrame:
577     """
578     Create a DataFrame from PDF file information.
579
580     Args:
581         pdf_files: List of dictionaries with PDF file information
582
583     Returns:
584         DataFrame with PDF file information
585     """
586     return pd.DataFrame(pdf_files)
587
588 def extract_text_from_pdf(pdf_path: str) -> str:
589     """
590     Extract text from a PDF file.
591
592     Args:
593         pdf_path: Path to the PDF file
594
595     Returns:
596         Extracted text
597     """
598     logger.info(f"Extracting text from: {pdf_path}")
599
600     try:
601         with fitz.open(pdf_path) as doc:
602             text = ""
603             for page_num, page in enumerate(doc):
604                 # Get text with blocks (preserves some structure)
605                 text += page.get_text("blocks")
606                 text += "\n\n" # Add separation between pages
607
608             return text
609     except Exception as e:
610         logger.error(f"Error extracting text from {pdf_path}: {str(e)}")
611         return ""
612
613 def process_pdfs(directory_path: str) -> pd.DataFrame:
614     """

```

```

612     Process all PDFs in a directory.
613
614     Args:
615         directory_path: Path to the directory containing PDF files
616
617     Returns:
618         DataFrame with PDF information and extracted text
619     """
620     # Scan directory for PDFs
621     pdf_files = scan_directory(directory_path)
622
623     # Create DataFrame
624     df = create_pdf_dataframe(pdf_files)
625
626     # Extract text from each PDF
627     tqdm.pandas(desc="Extracting text")
628     df['text'] = df['path'].progress_apply(extract_text_from_pdf)
629
630     # Filter out PDFs with no text
631     text_lengths = df['text'].str.len()
632     logger.info(f"Text extraction statistics: min={text_lengths.min()},
633               max={text_lengths.max()}, "
634               f"mean={text_lengths.mean():.2f}, median={text_lengths.median()}")
635
636     empty_pdfs = df[df['text'].str.len() == 0]
637     if not empty_pdfs.empty:
638         logger.warning(f"Found {len(empty_pdfs)} PDFs with no extractable text")
639
640     return df
641
642 if __name__ == "__main__":
643     # Example usage
644     import sys
645     from pathlib import Path
646
647     # Add the project root to the Python path
648     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
649
650     from app.config.settings import PDF_UPLOAD_FOLDER
651
652     df = process_pdfs(PDF_UPLOAD_FOLDER)
653     print(f"Processed {len(df)} PDFs")
654     print(df.head())

```

3. Create a text chunking utility:

```
touch app/utils/text_chunking.py
```

4. Add the following to app/utils/text\_chunking.py:

```

659 import re
660 import uuid
661 from typing import List, Dict, Any
662 import pandas as pd
663 from langchain.text_splitter import RecursiveCharacterTextSplitter
664 from tqdm import tqdm
665 import logging
666
667 # Configure logging
668 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
669             %(message)s')
670 logger = logging.getLogger(__name__)
671
672 def clean_text(text: str) -> str:
673     """
674     Clean text by removing excessive whitespace and normalizing line breaks.
675
676     Args:
677         text: Raw text to clean
678
679     Returns:
680         Cleaned text
681     """
682     # Replace multiple line breaks with a single one
683     text = re.sub(r'\n{3,}', '\n\n', text)
684
685     # Replace multiple spaces with a single one
686     text = re.sub(r' {2,}', ' ', text)
687
688     # Strip whitespace from beginning and end
689     text = text.strip()
690
691     return text
692
693 def chunk_text(text: str, chunk_size: int = 500, chunk_overlap: int = 50) -> List[str]:
694     """
695     Split text into chunks using LangChain's RecursiveCharacterTextSplitter.
696
697     Args:
698         text: Text to split into chunks
699         chunk_size: Target size of each chunk
700         chunk_overlap: Overlap between chunks
701
702     Returns:
703         List of text chunks
704     """
705     # Clean the text first
706     text = clean_text(text)

```

```

706     # Create text splitter
707     text_splitter = RecursiveCharacterTextSplitter(
708         chunk_size=chunk_size,
709         chunk_overlap=chunk_overlap,
710         length_function=len,
711         separators=["\n\n", "\n", ". ", " ", ""]
712     )
713
714     # Split text into chunks
715     chunks = text_splitter.split_text(text)
716
717     return chunks
718
719 def process_chunks(df: pd.DataFrame, chunk_size: int = 500, chunk_overlap: int = 50,
720                  max_chunks_per_doc: int = 1000) -> pd.DataFrame:
721     """
722     Process a DataFrame of PDFs and split text into chunks.
723
724     Args:
725         df: DataFrame with PDF information and extracted text
726         chunk_size: Target size of each chunk
727         chunk_overlap: Overlap between chunks
728         max_chunks_per_doc: Maximum number of chunks per document
729
730     Returns:
731         DataFrame with text chunks
732     """
733     logger.info(f"Processing chunks with size={chunk_size}, overlap={chunk_overlap}")
734
735     chunks_data = []
736
737     for _, row in tqdm(df.iterrows(), total=len(df), desc="Chunking documents"):
738         # Skip if no text
739         if not row['text'] or len(row['text']) == 0:
740             continue
741
742         # Get chunks
743         chunks = chunk_text(row['text'], chunk_size, chunk_overlap)
744
745         # Limit chunks if necessary
746         if len(chunks) > max_chunks_per_doc:
747             logger.warning(f"Document {row['filename']} has {len(chunks)} chunks, "
748                           f"limiting to {max_chunks_per_doc}")
749             chunks = chunks[:max_chunks_per_doc]
750
751         # Add chunks to list
752         for i, chunk_text in enumerate(chunks):
753             chunk_data = {
754                 'chunk_id': str(uuid.uuid4()),
755                 'pdf_path': row['path'],

```



```

753         'filename': row['filename'],
754         'chunk_index': i,
755         'chunk_text': chunk_text,
756         'token_count': len(chunk_text.split())
757     }
758     chunks_data.append(chunk_data)
759
760     # Create DataFrame from chunks
761     chunks_df = pd.DataFrame(chunks_data)
762
763     logger.info(f"Created {len(chunks_df)} chunks from {len(df)} documents")
764
765     return chunks_df
766
767 if __name__ == "__main__":
768     # Example usage
769     import sys
770     from pathlib import Path
771
772     # Add the project root to the Python path
773     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
774
775     from app.config.settings import PDF_UPLOAD_FOLDER, CHUNK_SIZE, CHUNK_OVERLAP,
776     MAX_CHUNKS_PER_DOC
777     from app.utils.pdf_ingestion import process_pdfs
778
779     # Process PDFs
780     pdf_df = process_pdfs(PDF_UPLOAD_FOLDER)
781
782     # Process chunks
783     chunks_df = process_chunks(pdf_df, CHUNK_SIZE, CHUNK_OVERLAP, MAX_CHUNKS_PER_DOC)
784
785     print(f"Created {len(chunks_df)} chunks")
786     print(chunks_df.head())
787
788
789

```

**Professor's Hint:** When extracting text from PDFs, remember that they're essentially containers of independent objects rather than structured documents. Many PDFs, especially scientific papers with multiple columns, can be challenging to extract in reading order. Consider extracting "blocks" (as shown in the code) for a balance between structure preservation and accuracy.

5. Create a test script for PDF processing:

```
touch app/tests/test_pdf_processing.py
```

6. Add the following to app/tests/test\_pdf\_processing.py:

```

import os
import sys
import pytest
from pathlib import Path

```

```

800
801 # Add the project root to the Python path
802 sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
803
804 from app.utils.pdf_ingestion import scan_directory, extract_text_from_pdf, process_pdfs
805 from app.utils.text_chunking import chunk_text, process_chunks
806 from app.config.settings import PDF_UPLOAD_FOLDER, CHUNK_SIZE, CHUNK_OVERLAP
807
808 def test_scan_directory():
809     """Test scanning directory for PDFs."""
810     # Create a test PDF if none exists
811     if not list(Path(PDF_UPLOAD_FOLDER).rglob('*.pdf')):
812         pytest.skip("No PDF files found for testing")
813
814     pdfs = scan_directory(PDF_UPLOAD_FOLDER)
815     assert len(pdfs) > 0, "No PDFs found in directory"
816     assert 'path' in pdfs[0], "PDF info missing 'path'"
817     assert 'filename' in pdfs[0], "PDF info missing 'filename'"
818
819 def test_extract_text():
820     """Test extracting text from a PDF."""
821     # Create a test PDF if none exists
822     if not list(Path(PDF_UPLOAD_FOLDER).rglob('*.pdf')):
823         pytest.skip("No PDF files found for testing")
824
825     pdf_path = next(Path(PDF_UPLOAD_FOLDER).rglob('*.pdf'))
826     text = extract_text_from_pdf(str(pdf_path))
827     assert text, "No text extracted from PDF"
828
829 def test_chunk_text():
830     """Test chunking text."""
831     sample_text = """
832     This is a sample document that will be split into chunks.
833     It has multiple sentences and paragraphs.
834
835     This is the second paragraph with some more text.
836     We want to make sure the chunking works correctly.
837
838     Let's add a third paragraph to ensure we have enough text to create multiple chunks.
839     This should be enough for the test.
840     """
841
842     chunks = chunk_text(sample_text, chunk_size=100, chunk_overlap=20)
843     assert len(chunks) > 1, "Text not split into multiple chunks"
844     assert len(chunks[0]) <= 100 + 20, "Chunk size exceeds expected maximum"
845
846 if __name__ == "__main__":
847     # Run tests
848     pytest.main(["-xvs", __file__])

```

7. Run the test script:

```
python app/tests/test_pdf_processing.py
```

#### Checkpoint Questions:

1. What are some challenges with extracting text from PDFs?
2. Why do we use chunking with overlap instead of simply splitting the text at fixed intervals?
3. How would the choice of `chunk_size` and `chunk_overlap` impact the RAG system?
4. How do different PDF extraction techniques handle multi-column layouts, tables, and graphics?
5. What architectural changes would be needed to handle non-PDF documents like Word, PowerPoint, or HTML?
6. How might language-specific considerations affect the text extraction and chunking process for multilingual documents?
7. **Additional Challenge:** Enhance the PDF extraction to handle tables and preserve their structure using PyMuPDF's table extraction capabilities.

#### Lab 2.2: Vector Database Setup and Embedding Generation.

##### Learning Objectives:

- Implement vector embedding generation
- Set up a vector database for semantic search
- Design an efficient document storage system

##### Exercises:

1. Create an embedding utility:

```
touch app/utils/embedding_generation.py
```

2. Add the following to `app/utils/embedding_generation.py`:

```
import os
import numpy as np
import pandas as pd
from typing import List, Dict, Any
from sentence_transformers import SentenceTransformer
from tqdm import tqdm
import logging

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

class EmbeddingGenerator:
    def __init__(self, model_path: str, batch_size: int = 32):
        """
        Initialize the embedding generator.

        Args:
            model_path: Path to the SentenceTransformer model.
            batch_size: Batch size for embedding generation.
        """
```

```

894         model_path: Path to the embedding model
895         batch_size: Batch size for embedding generation
896     """
897     logger.info(f"Loading embedding model from {model_path}")
898     self.model = SentenceTransformer(model_path)
899     self.batch_size = batch_size
900     self.embedding_dim = self.model.get_sentence_embedding_dimension()
901     logger.info(f"Model loaded with embedding dimension: {self.embedding_dim}")
902
903 def generate_embeddings(self, texts: List[str]) -> np.ndarray:
904     """
905     Generate embeddings for a list of texts.
906
907     Args:
908         texts: List of texts to embed
909
910     Returns:
911         Array of embeddings
912     """
913     logger.info(f"Generating embeddings for {len(texts)} texts with batch size
914 {self.batch_size}")
915
916     embeddings = self.model.encode(
917         texts,
918         batch_size=self.batch_size,
919         show_progress_bar=True,
920         convert_to_numpy=True,
921         normalize_embeddings=True # L2 normalization for cosine similarity
922     )
923
924     logger.info(f"Generated embeddings with shape: {embeddings.shape}")
925     return embeddings
926
927 def process_dataframe(self, df: pd.DataFrame, text_column: str = 'chunk_text',
928                     embedding_column: str = 'embedding') -> pd.DataFrame:
929     """
930     Process a DataFrame and add embeddings.
931
932     Args:
933         df: DataFrame with text chunks
934         text_column: Name of the column containing text
935         embedding_column: Name of the column to store embeddings
936
937     Returns:
938         DataFrame with embeddings
939     """
940     logger.info(f"Processing DataFrame with {len(df)} rows")
941
942     # Get texts
943     texts = df[text_column].tolist()

```

```

941
942     # Generate embeddings
943     embeddings = self.generate_embeddings(texts)
944
945     # Add embeddings to DataFrame
946     df[embedding_column] = list(embeddings)
947
948     return df
949
950 def embed_chunks(chunks_df: pd.DataFrame, model_path: str, batch_size: int = 32) ->
951     pd.DataFrame:
952     """
953     Generate embeddings for text chunks.
954
955     Args:
956         chunks_df: DataFrame with text chunks
957         model_path: Path to the embedding model
958         batch_size: Batch size for embedding generation
959
960     Returns:
961         DataFrame with embeddings
962     """
963     # Create embedder
964     embedder = EmbeddingGenerator(model_path, batch_size)
965
966     # Process DataFrame
967     chunks_df = embedder.process_dataframe(chunks_df)
968
969     return chunks_df
970
971 if __name__ == "__main__":
972     # Example usage
973     import sys
974     from pathlib import Path
975
976     # Add the project root to the Python path
977     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
978
979     from app.config.settings import PDF_UPLOAD_FOLDER, EMBEDDING_MODEL_PATH, CHUNK_SIZE,
980     CHUNK_OVERLAP
981     from app.utils.pdf_ingestion import process_pdfs
982     from app.utils.text_chunking import process_chunks
983
984     # Process PDFs
985     pdf_df = process_pdfs(PDF_UPLOAD_FOLDER)
986
987     # Process chunks
988     chunks_df = process_chunks(pdf_df, CHUNK_SIZE, CHUNK_OVERLAP)
989
990     # Generate embeddings

```

```

988     chunks_with_embeddings = embed_chunks(chunks_df, EMBEDDING_MODEL_PATH)
989
990     print(f"Generated embeddings for {len(chunks_with_embeddings)} chunks")
991     print(f"Embedding dimension: {len(chunks_with_embeddings['embedding'].iloc[0])}")

```

### 3. Create a vector database client:

```
touch app/utils/vector_db.py
```

### 4. Add the following to app/utils/vector\_db.py:

```

998 import os
999 import pandas as pd
1000 import numpy as np
1001 from typing import List, Dict, Any, Optional, Tuple
1002 from qdrant_client import QdrantClient
1003 from qdrant_client.http import models
1004 from tqdm import tqdm
1005 import logging
1006
1007 # Configure logging
1008 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
1009     %(message)s')
1010 logger = logging.getLogger(__name__)
1011
1012 class VectorDBClient:
1013     def __init__(self, host: str, port: int, collection_name: str, vector_size: int):
1014         """
1015         Initialize the vector database client.
1016
1017         Args:
1018             host: Host of the Qdrant server
1019             port: Port of the Qdrant server
1020             collection_name: Name of the collection to use
1021             vector_size: Dimension of the embedding vectors
1022         """
1023         logger.info(f"Connecting to Qdrant at {host}:{port}")
1024         self.client = QdrantClient(host=host, port=port)
1025         self.collection_name = collection_name
1026         self.vector_size = vector_size
1027
1028     def create_collection(self) -> None:
1029         """Create a collection in the vector database."""
1030         logger.info(f"Creating collection: {self.collection_name}")
1031
1032         # Check if collection already exists
1033         collections = self.client.get_collections().collections
1034         collection_names = [collection.name for collection in collections]
1035
1036         if self.collection_name in collection_names:

```

```

1035         logger.info(f"Collection {self.collection_name} already exists")
1036         return
1037
1038     # Create collection
1039     self.client.create_collection(
1040         collection_name=self.collection_name,
1041         vectors_config=models.VectorParams(
1042             size=self.vector_size,
1043             distance=models.Distance.COSINE
1044         ),
1045         # Add optimizers config for better performance
1046         optimizers_config=models.OptimizersConfigDiff(
1047             memmap_threshold=20000 # Use memmapped storage for collections > 20k vectors
1048         )
1049     )
1050
1051     logger.info(f"Collection {self.collection_name} created")
1052
1053     def delete_collection(self) -> None:
1054         """Delete the collection."""
1055         logger.info(f"Deleting collection: {self.collection_name}")
1056         try:
1057             self.client.delete_collection(collection_name=self.collection_name)
1058             logger.info(f"Collection {self.collection_name} deleted")
1059         except Exception as e:
1060             logger.error(f"Error deleting collection: {str(e)}")
1061
1062     def upload_vectors(self, df: pd.DataFrame,
1063                       vector_column: str = 'embedding',
1064                       batch_size: int = 100) -> None:
1065         """
1066         Upload vectors to the collection.
1067
1068         Args:
1069             df: DataFrame with embeddings
1070             vector_column: Name of the column containing embeddings
1071             batch_size: Batch size for uploading
1072         """
1073         logger.info(f"Uploading {len(df)} vectors to collection {self.collection_name}")
1074
1075         # Ensure collection exists
1076         self.create_collection()
1077
1078         # Prepare points for upload
1079         points = []
1080
1081         for i, row in tqdm(df.iterrows(), total=len(df), desc="Preparing vectors"):
1082             # Convert embedding to list if it's a numpy array
1083             embedding = row[vector_column]
1084             if isinstance(embedding, np.ndarray):

```

```

1082         embedding = embedding.tolist()
1083
1084         # Create point
1085         point = models.PointStruct(
1086             id=i, # Use DataFrame index as ID
1087             vector=embedding,
1088             payload={
1089                 'chunk_id': row['chunk_id'],
1090                 'pdf_path': row['pdf_path'],
1091                 'filename': row['filename'],
1092                 'chunk_index': row['chunk_index'],
1093                 'chunk_text': row['chunk_text'],
1094                 'token_count': row['token_count']
1095             }
1096         )
1097         points.append(point)
1098
1099         # Upload in batches
1100         total_batches = (len(points) + batch_size - 1) // batch_size
1101         for i in tqdm(range(0, len(points), batch_size), total=total_batches, desc="Uploading
batches"):
1102             batch = points[i:i+batch_size]
1103             self.client.upsert(
1104                 collection_name=self.collection_name,
1105                 points=batch
1106             )
1107
1108             logger.info(f"Uploaded {len(df)} vectors to collection {self.collection_name}")
1109
1110     def search(self, query_vector: List[float], limit: int = 5) -> List[Dict]:
1111         """
1112         Search for similar vectors.
1113
1114         Args:
1115             query_vector: Query vector
1116             limit: Maximum number of results
1117
1118         Returns:
1119             List of search results
1120         """
1121         logger.info(f"Searching collection {self.collection_name} for similar vectors")
1122
1123         # Convert query vector to list if it's a numpy array
1124         if isinstance(query_vector, np.ndarray):
1125             query_vector = query_vector.tolist()
1126
1127         # Search
1128         results = self.client.search(
1129             collection_name=self.collection_name,

```



```

1129         query_vector=query_vector,
1130         limit=limit
1131     )
1132
1133     # Convert to list of dictionaries
1134     search_results = []
1135     for result in results:
1136         item = result.payload
1137         item['score'] = result.score
1138         search_results.append(item)
1139
1140     logger.info(f"Found {len(search_results)} results")
1141     return search_results
1142
1143 def count_vectors(self) -> int:
1144     """
1145     Count the number of vectors in the collection.
1146
1147     Returns:
1148         Number of vectors
1149     """
1150     try:
1151         count = self.client.count(collection_name=self.collection_name).count
1152         return count
1153     except Exception as e:
1154         logger.error(f"Error counting vectors: {str(e)}")
1155         return 0
1156
1157 def setup_vector_db(host: str, port: int, collection_name: str, vector_size: int) ->
1158     VectorDBClient:
1159     """
1160     Set up the vector database.
1161
1162     Args:
1163         host: Host of the Qdrant server
1164         port: Port of the Qdrant server
1165         collection_name: Name of the collection to use
1166         vector_size: Dimension of the embedding vectors
1167
1168     Returns:
1169         Vector database client
1170     """
1171     # Create client
1172     client = VectorDBClient(host, port, collection_name, vector_size)
1173
1174     # Create collection
1175     client.create_collection()
1176
1177     return client

```

```

1176 if __name__ == "__main__":
1177     # Example usage
1178     import sys
1179     from pathlib import Path
1180
1181     # Add the project root to the Python path
1182     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
1183
1184     from app.config.settings import (
1185         VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
1186         PDF_UPLOAD_FOLDER, EMBEDDING_MODEL_PATH, CHUNK_SIZE, CHUNK_OVERLAP
1187     )
1188     from app.utils.pdf_ingestion import process_pdfs
1189     from app.utils.text_chunking import process_chunks
1190     from app.utils.embedding_generation import embed_chunks
1191
1192     # Process PDFs
1193     pdf_df = process_pdfs(PDF_UPLOAD_FOLDER)
1194
1195     # Process chunks
1196     chunks_df = process_chunks(pdf_df, CHUNK_SIZE, CHUNK_OVERLAP)
1197
1198     # Generate embeddings
1199     chunks_with_embeddings = embed_chunks(chunks_df, EMBEDDING_MODEL_PATH)
1200
1201     # Set up vector database
1202     vector_db = setup_vector_db(VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME,
1203                                VECTOR_DIMENSION)
1204
1205     # Upload vectors
1206     vector_db.upload_vectors(chunks_with_embeddings)
1207
1208     # Count vectors
1209     count = vector_db.count_vectors()
1210     print(f"Vector database contains {count} vectors")

```

**Professor's Hint:** Vector database performance is critical for RAG applications. Qdrant offers excellent performance with minimal resource usage, making it suitable for our MacMini deployment. The cosine distance metric is used because our embeddings are normalized, making it equivalent to dot product but slightly more intuitive a score of 1.0 means perfect similarity.

5. Create a pipeline script to orchestrate the entire process:

```
touch app/pipeline.py
```

6. Add the following to app/pipeline.py:

```

import os
import sys
import argparse

```

```

1223 import logging
1224 from pathlib import Path
1225
1226 # Configure logging
1227 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
1228             %(message)s')
1229 logger = logging.getLogger(__name__)
1230
1231 # Add the project root to the Python path
1232 sys.path.insert(0, str(Path(__file__).resolve().parent.parent))
1233
1234 from app.config.settings import (
1235     PDF_UPLOAD_FOLDER, EMBEDDING_MODEL_PATH, CHUNK_SIZE, CHUNK_OVERLAP, MAX_CHUNKS_PER_DOC,
1236     VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION
1237 )
1238 from app.utils.pdf_ingestion import process_pdfs
1239 from app.utils.text_chunking import process_chunks
1240 from app.utils.embedding_generation import embed_chunks
1241 from app.utils.vector_db import setup_vector_db
1242
1243 def run_pipeline(pdf_dir: str, rebuild_index: bool = False):
1244     """
1245     Run the full pipeline from PDF ingestion to vector database upload.
1246
1247     Args:
1248         pdf_dir: Directory containing PDF files
1249         rebuild_index: Whether to rebuild the vector index (delete and recreate)
1250     """
1251     logger.info(f"Starting pipeline with PDF directory: {pdf_dir}")
1252
1253     # Step 1: Process PDFs
1254     logger.info("Step 1: Processing PDFs")
1255     pdf_df = process_pdfs(pdf_dir)
1256     logger.info(f"Processed {len(pdf_df)} PDFs")
1257
1258     # Step 2: Process chunks
1259     logger.info("Step 2: Processing chunks")
1260     chunks_df = process_chunks(pdf_df, CHUNK_SIZE, CHUNK_OVERLAP, MAX_CHUNKS_PER_DOC)
1261     logger.info(f"Created {len(chunks_df)} chunks")
1262
1263     # Step 3: Generate embeddings
1264     logger.info("Step 3: Generating embeddings")
1265     chunks_with_embeddings = embed_chunks(chunks_df, EMBEDDING_MODEL_PATH)
1266     logger.info(f"Generated embeddings for {len(chunks_with_embeddings)} chunks")
1267
1268     # Step 4: Set up vector database
1269     logger.info("Step 4: Setting up vector database")
1270     vector_db = setup_vector_db(VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME,
1271                                VECTOR_DIMENSION)

```

```

1270     # Delete collection if rebuilding index
1271     if rebuild_index:
1272         logger.info("Rebuilding vector index: deleting existing collection")
1273         vector_db.delete_collection()
1274         vector_db.create_collection()
1275
1276     # Step 5: Upload vectors
1277     logger.info("Step 5: Uploading vectors")
1278     vector_db.upload_vectors(chunks_with_embeddings)
1279
1280     # Verify upload
1281     count = vector_db.count_vectors()
1282     logger.info(f"Pipeline complete. Vector database contains {count} vectors")
1283
1284 if __name__ == "__main__":
1285     # Parse arguments
1286     parser = argparse.ArgumentParser(description='Run the PDF processing pipeline')
1287     parser.add_argument('--pdf-dir', type=str, default=PDF_UPLOAD_FOLDER,
1288                         help='Directory containing PDF files')
1289     parser.add_argument('--rebuild', action='store_true',
1290                         help='Rebuild the vector index (delete and recreate)')
1291     args = parser.parse_args()
1292
1293     # Run pipeline
1294     run_pipeline(args.pdf_dir, args.rebuild)
1295
1296
1297
1298
1299

```

7. Create a test script for the vector database:

```
touch app/tests/test_vector_db.py
```

8. Add the following to app/tests/test\_vector\_db.py:

```

1299 import os
1300 import sys
1301 import pytest
1302 import numpy as np
1303 from pathlib import Path
1304
1305 # Add the project root to the Python path
1306 sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
1307
1308 from app.utils.vector_db import VectorDBClient
1309 from app.config.settings import VECTOR_DB_HOST, VECTOR_DB_PORT, VECTOR_DIMENSION
1310
1311 def test_vector_db_connection():
1312     """Test connecting to the vector database."""
1313     # Create client
1314     client = VectorDBClient(VECTOR_DB_HOST, VECTOR_DB_PORT, "test_collection",
1315                             VECTOR_DIMENSION)
1316
1317
1318
1319
1320

```

```

1317     # Check connection
1318     try:
1319         collections = client.client.get_collections()
1320         assert collections is not None, "Failed to get collections"
1321     except Exception as e:
1322         pytest.fail(f"Failed to connect to vector database: {str(e)}")
1323
1324 def test_collection_operations():
1325     """Test collection operations."""
1326     # Create client
1327     client = VectorDBClient(VECTOR_DB_HOST, VECTOR_DB_PORT, "test_collection",
1328                             VECTOR_DIMENSION)
1329
1330     # Create collection
1331     client.create_collection()
1332
1333     # Check if collection exists
1334     collections = client.client.get_collections().collections
1335     collection_names = [collection.name for collection in collections]
1336     assert "test_collection" in collection_names, "Collection not created"
1337
1338     # Delete collection
1339     client.delete_collection()
1340
1341     # Check if collection is deleted
1342     collections = client.client.get_collections().collections
1343     collection_names = [collection.name for collection in collections]
1344     assert "test_collection" not in collection_names, "Collection not deleted"
1345
1346 def test_vector_operations():
1347     """Test vector operations."""
1348     # Create client
1349     client = VectorDBClient(VECTOR_DB_HOST, VECTOR_DB_PORT, "test_vectors", VECTOR_DIMENSION)
1350
1351     # Create collection and clear any existing data
1352     client.delete_collection()
1353     client.create_collection()
1354
1355     # Create test vectors
1356     import pandas as pd
1357
1358     # Create 10 random test vectors
1359     np.random.seed(42) # For reproducibility
1360     test_vectors = []
1361     for i in range(10):
1362         vec = np.random.rand(VECTOR_DIMENSION)
1363         # Normalize for cosine similarity
1364         vec = vec / np.linalg.norm(vec)
1365         test_vectors.append(vec)

```

```

1364 # Create test dataframe
1365 df = pd.DataFrame({
1366     'chunk_id': [f"chunk_{i}" for i in range(10)],
1367     'pdf_path': [f"/path/to/pdf_{i}.pdf" for i in range(10)],
1368     'filename': [f"pdf_{i}.pdf" for i in range(10)],
1369     'chunk_index': list(range(10)),
1370     'chunk_text': [f"This is test chunk {i}" for i in range(10)],
1371     'token_count': [len(f"This is test chunk {i}".split()) for i in range(10)],
1372     'embedding': test_vectors
1373 })
1374 # Upload vectors
1375 client.upload_vectors(df)
1376
1377 # Check if vectors are uploaded
1378 count = client.count_vectors()
1379 assert count == 10, f"Expected 10 vectors, got {count}"
1380
1381 # Search for similar vector
1382 results = client.search(test_vectors[0])
1383 assert len(results) > 0, "No search results returned"
1384 assert results[0]['chunk_id'] == "chunk_0", "First result should be the query vector
1385 itself"
1386
1387 # Clean up
1388 client.delete_collection()
1389
1390 if __name__ == "__main__":
1391     # Run tests
1392     pytest.main(["-xvs", __file__])

```

### Checkpoint Questions:

1. Why do we normalize embeddings before storing them in the vector database?
2. What are the tradeoffs between different distance metrics (cosine, Euclidean, dot product)?
3. How does batch processing improve performance when generating embeddings or uploading to the vector database?
4. How would you modify the embedding strategy if you needed to handle documents in multiple languages?
5. How does the choice of vector dimension affect the semantic richness versus computational efficiency tradeoff?
6. What information might be lost during the chunking process, and how could this affect retrieval quality?
7. **Additional Challenge:** Implement a method to incrementally update the vector database when new PDFs are added, without reprocessing the entire corpus.

## Lab 3: Query Processing and RAG Implementation

### Lab 3.1: Vector Search and Re-ranking.

*Learning Objectives:*

- Implement effective query processing
- Build a re-ranking system for search results
- Optimize search relevance using modern techniques

*Exercises:*

1. Create a query processing utility:

```
touch app/utils/query_processing.py
```

2. Add the following to app/utils/query\_processing.py:

```
import os
import numpy as np
from typing import List, Dict, Any
from sentence_transformers import SentenceTransformer
import logging

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

class QueryProcessor:
    def __init__(self, model_path: str):
        """
        Initialize the query processor.

        Args:
            model_path: Path to the embedding model

        """
        logger.info(f"Loading embedding model from {model_path}")
        self.model = SentenceTransformer(model_path)
        self.embedding_dim = self.model.get_sentence_embedding_dimension()
        logger.info(f"Model loaded with embedding dimension: {self.embedding_dim}")

    def process_query(self, query: str) -> np.ndarray:
        """
        Process a query and generate an embedding.

        Args:
            query: Query text

        Returns:
            Query embedding

        """
        logger.info(f"Processing query: {query}")

        # Generate embedding
```

```

1458         embedding = self.model.encode(
1459             query,
1460             convert_to_numpy=True,
1461             normalize_embeddings=True # L2 normalization for cosine similarity
1462         )
1463
1464         logger.info(f"Generated query embedding with shape: {embedding.shape}")
1465         return embedding
1466
1467 def process_query(query: str, model_path: str) -> np.ndarray:
1468     """
1469     Process a query and generate an embedding.
1470
1471     Args:
1472         query: Query text
1473         model_path: Path to the embedding model
1474
1475     Returns:
1476         Query embedding
1477     """
1478     processor = QueryProcessor(model_path)
1479     return processor.process_query(query)
1480
1481 if __name__ == "__main__":
1482     # Example usage
1483     import sys
1484     from pathlib import Path
1485
1486     # Add the project root to the Python path
1487     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
1488
1489     from app.config.settings import EMBEDDING_MODEL_PATH
1490
1491     # Process a query
1492     query = "What is retrieval-augmented generation?"
1493     embedding = process_query(query, EMBEDDING_MODEL_PATH)
1494     print(f"Query embedding shape: {embedding.shape}")

```

### 3. Create a re-ranking utility:

```
touch app/utils/reranking.py
```

### 4. Add the following to app/utils/reranking.py:

```

import os
import numpy as np
from typing import List, Dict, Any
from sentence_transformers import CrossEncoder
import logging

```



```

1505 # Configure logging
1506 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
1507     %(message)s')
1508 logger = logging.getLogger(__name__)
1509
1510 class Reranker:
1511     def __init__(self, model_path: str):
1512         """
1513         Initialize the reranker.
1514
1515         Args:
1516             model_path: Path to the reranker model
1517         """
1518         logger.info(f"Loading reranker model from {model_path}")
1519         self.model = CrossEncoder(model_path, max_length=512)
1520         logger.info("Reranker model loaded")
1521
1522     def rerank(self, query: str, results: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
1523         """
1524         Rerank search results.
1525
1526         Args:
1527             query: Query text
1528             results: List of search results
1529
1530         Returns:
1531             Reranked results
1532         """
1533         logger.info(f"Reranking {len(results)} results for query: {query}")
1534
1535         if not results:
1536             return []
1537
1538         # Create pairs for the cross-encoder
1539         pairs = [(query, result['chunk_text']) for result in results]
1540
1541         # Get scores
1542         scores = self.model.predict(pairs)
1543
1544         # Add scores to results
1545         for i, score in enumerate(scores):
1546             results[i]['rerank_score'] = float(score)
1547
1548         # Sort by rerank score
1549         reranked_results = sorted(results, key=lambda x: x['rerank_score'], reverse=True)
1550
1551         logger.info("Reranking complete")
1552         return reranked_results

```

```

1552 def rerank_results(query: str, results: List[Dict[str, Any]], model_path: str) ->
1553     List[Dict[str, Any]]:
1554     """
1555     Rerank search results.
1556
1557     Args:
1558         query: Query text
1559         results: List of search results
1560         model_path: Path to the reranker model
1561
1562     Returns:
1563         Reranked results
1564     """
1565     reranker = Reranker(model_path)
1566     return reranker.rerank(query, results)
1567
1568 if __name__ == "__main__":
1569     # Example usage
1570     import sys
1571     from pathlib import Path
1572
1573     # Add the project root to the Python path
1574     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
1575
1576     from app.config.settings import (
1577         RERANKER_MODEL_PATH, EMBEDDING_MODEL_PATH,
1578         VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION
1579     )
1580     from app.utils.query_processing import process_query
1581     from app.utils.vector_db import VectorDBClient
1582
1583     # Process a query
1584     query = "What is retrieval-augmented generation?"
1585     embedding = process_query(query, EMBEDDING_MODEL_PATH)
1586
1587     # Search the vector database
1588     vector_db = VectorDBClient(VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME,
1589                               VECTOR_DIMENSION)
1590     results = vector_db.search(embedding, limit=10)
1591
1592     # Rerank results
1593     reranked_results = rerank_results(query, results, RERANKER_MODEL_PATH)
1594
1595     # Print results
1596     print("Reranked results:")
1597     for i, result in enumerate(reranked_results[:5]):
1598         print(f"{i+1}. Score: {result['rerank_score']:.4f}, Original Score:

```

**Professor's Hint:** *Re-ranking is one of the most effective yet underutilized techniques in RAG systems. While vector similarity gives us a “ballpark” match, cross-encoders consider the query and document together to produce much more accurate relevance scores. The computational cost is higher, but since we only re-rank a small number of results, the overall impact is minimal.*

5. Create a search pipeline that combines vector search and re-ranking:

```
touch app/utils/search.py
```

6. Add the following to `app/utils/search.py`:

```
import os
from typing import List, Dict, Any
import logging

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

class SearchPipeline:
    def __init__(self, vector_db_client, query_processor, reranker,
                 max_results: int = 10, rerank_results: int = 10):
        """
        Initialize the search pipeline.

        Args:
            vector_db_client: Vector database client
            query_processor: Query processor
            reranker: Reranker
            max_results: Maximum number of results to return
            rerank_results: Number of results to rerank
        """
        self.vector_db_client = vector_db_client
        self.query_processor = query_processor
        self.reranker = reranker
        self.max_results = max_results
        self.rerank_results = rerank_results

    def search(self, query: str) -> List[Dict[str, Any]]:
        """
        Search for documents relevant to a query.

        Args:
            query: Query text

        Returns:
            Search results
        """
        logger.info(f"Searching for: {query}")
```

```

1646
1647     # Process query
1648     query_embedding = self.query_processor.process_query(query)
1649
1650     # Search vector database
1651     vector_results = self.vector_db_client.search(query_embedding,
1652 limit=self.rerank_results)
1653     logger.info(f"Found {len(vector_results)} results from vector search")
1654
1655     if not vector_results:
1656         logger.warning("No results found in vector search")
1657         return []
1658
1659     # Rerank results
1660     reranked_results = self.reranker.rerank(query, vector_results)
1661     logger.info("Results reranked")
1662
1663     # Return top results
1664     return reranked_results[:self.max_results]
1665
1666 def create_search_pipeline(vector_db_host: str, vector_db_port: int, collection_name: str,
1667 vector_dimension: int, embedding_model_path: str,
1668 reranker_model_path: str,
1669 max_results: int = 5, rerank_top_k: int = 10):
1670
1671     """
1672     Create a search pipeline.
1673
1674     Args:
1675         vector_db_host: Host of the vector database
1676         vector_db_port: Port of the vector database
1677         collection_name: Name of the collection
1678         vector_dimension: Dimension of the embedding vectors
1679         embedding_model_path: Path to the embedding model
1680         reranker_model_path: Path to the reranker model
1681         max_results: Maximum number of results to return
1682         rerank_top_k: Number of results to rerank
1683
1684     Returns:
1685         Search pipeline
1686     """
1687
1688     # Import here to avoid circular imports
1689     from app.utils.vector_db import VectorDBClient
1690     from app.utils.query_processing import QueryProcessor
1691     from app.utils.reranking import Reranker
1692
1693     # Create components
1694     vector_db_client = VectorDBClient(vector_db_host, vector_db_port, collection_name,
1695 vector_dimension)
1696     query_processor = QueryProcessor(embedding_model_path)
1697     reranker = Reranker(reranker_model_path)

```

```

1693
1694 # Create pipeline
1695 pipeline = SearchPipeline(
1696     vector_db_client, query_processor, reranker,
1697     max_results, rerank_top_k
1698 )
1699
1700 return pipeline
1701
1702 if __name__ == "__main__":
1703     # Example usage
1704     import sys
1705     from pathlib import Path
1706
1707     # Add the project root to the Python path
1708     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
1709
1710     from app.config.settings import (
1711         VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
1712         EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH
1713     )
1714
1715     # Create search pipeline
1716     pipeline = create_search_pipeline(
1717         VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
1718         EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH
1719     )
1720
1721     # Search
1722     query = "What is retrieval-augmented generation?"
1723     results = pipeline.search(query)
1724
1725     # Print results
1726     print(f"Top {len(results)} results for query: {query}")
1727     for i, result in enumerate(results):
1728         print(f"{i+1}. Score: {result['rerank_score']:.4f}, Vector Score:
1729             {result['score']:.4f}")
1730         print(f"    File: {result['filename']}")
1731         print(f"    Text: {result['chunk_text'][:200]}...")
1732         print()

```

### Checkpoint Questions:

1. How does the two-stage retrieval process (vector search re-ranking) improve result quality?
2. What are the performance implications of increasing the number of results to re-rank?
3. Why is it important to normalize query embeddings in the same way as document embeddings?
4. How might different similarity thresholds affect recall versus precision in retrieval results?
5. What architectural changes would be needed if you wanted to incorporate hybrid search (combining vector similarity with keyword matching)?

6. How would the search pipeline need to be modified to support multi-modal queries (such as image + text)?

7. **Additional Challenge:** Implement a hybrid search that combines vector similarity with BM25 keyword matching for improved retrieval of rare terms and phrases.

### Lab 3.2: LLM Integration and Response Generation.

#### Learning Objectives:

- Integrate a local LLM for answer generation
- Design effective prompts for RAG systems
- Implement context augmentation techniques

#### Exercises:

1. Create an LLM utility:

```
touch app/utils/llm.py
```

2. Add the following to app/utils/llm.py:

```
import os
from typing import Dict, Any, List, Optional
from llama_cpp import Llama
import logging

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

class LLMProcessor:
    def __init__(self, model_path: str, context_size: int = 2048, max_tokens: int = 512):
        """
        Initialize the LLM processor.

        Args:
            model_path: Path to the LLM model
            context_size: Context size for the model
            max_tokens: Maximum number of tokens to generate
        """
        logger.info(f"Loading LLM from {model_path}")
        self.model = Llama(
            model_path=model_path,
            n_ctx=context_size,
            n_batch=512, # Adjust based on available RAM
        )
        self.max_tokens = max_tokens
        logger.info("LLM loaded")

    def create_prompt(self, query: str, context: List[Dict[str, Any]]) -> str:
        """
```

```

1787     Create a prompt for the LLM.
1788
1789     Args:
1790         query: User query
1791         context: List of context documents
1792
1793     Returns:
1794         Formatted prompt
1795     """
1796     # Format context
1797     context_text = ""
1798     for i, doc in enumerate(context):
1799         context_text += f"Document {i+1}: \n{doc['chunk_text']}\n\n"
1800
1801     # Create prompt for Llama 3
1802     prompt = f"""<|system|>
1803     You are a helpful AI assistant that provides accurate and concise answers based on the
1804     provided context documents.
1805     If the answer is not contained in the documents, say "I don't have enough information to
1806     answer this question."
1807     Do not make up or hallucinate any information that is not supported by the documents.
1808     </|system|>
1809
1810     <|user|>
1811     I need information about the following topic:
1812
1813     {query}
1814
1815     Here are relevant documents to help answer this question:
1816
1817     {context_text}
1818     </|user|>
1819
1820     <|assistant|>
1821     """
1822     return prompt
1823
1824     def generate_response(self, prompt: str) -> Dict[str, Any]:
1825         """
1826         Generate a response from the LLM.
1827
1828         Args:
1829             prompt: Formatted prompt
1830
1831         Returns:
1832             Response with text and metadata
1833         """
1834         logger.info("Generating LLM response")
1835
1836         # Generate response

```

```

1834         response = self.model(
1835             prompt,
1836             max_tokens=self.max_tokens,
1837             stop=["User query:", "\n\n"],
1838             temperature=0.2, # Lower temperature for more factual responses
1839             top_p=0.9,
1840             top_k=40,
1841             repeat_penalty=1.1
1842         )
1843
1844         # Extract text
1845         response_text = response['choices'][0]['text'].strip()
1846
1847         # Get metadata
1848         metadata = {
1849             'tokens_used': len(response['usage']['prompt_tokens']) +
1850             len(response['usage']['completion_tokens']),
1851             'prompt_tokens': len(response['usage']['prompt_tokens']),
1852             'completion_tokens': len(response['usage']['completion_tokens']),
1853         }
1854
1855         logger.info(f"Generated response with {metadata['completion_tokens']} tokens")
1856
1857         return {
1858             'text': response_text,
1859             'metadata': metadata
1860         }
1861
1862     class RAGProcessor:
1863     def __init__(self, search_pipeline, llm_processor):
1864         """
1865         Initialize the RAG processor.
1866
1867         Args:
1868             search_pipeline: Search pipeline
1869             llm_processor: LLM processor
1870         """
1871         self.search_pipeline = search_pipeline
1872         self.llm_processor = llm_processor
1873
1874     def process_query(self, query: str) -> Dict[str, Any]:
1875         """
1876         Process a query using RAG.
1877
1878         Args:
1879             query: User query
1880
1881         Returns:
1882             Response with text, sources, and metadata
1883         """

```



```

1881     logger.info(f"Processing RAG query: {query}")
1882
1883     # Search for relevant documents
1884     search_results = self.search_pipeline.search(query)
1885
1886     if not search_results:
1887         return {
1888             'text': "I couldn't find any relevant information to answer your question.",
1889             'sources': [],
1890             'metadata': {'search_results': 0}
1891         }
1892
1893     # Create prompt
1894     prompt = self.llm_processor.create_prompt(query, search_results)
1895
1896     # Generate response
1897     response = self.llm_processor.generate_response(prompt)
1898
1899     # Format sources
1900     sources = []
1901     for result in search_results:
1902         sources.append({
1903             'filename': result['filename'],
1904             'chunk_text': result['chunk_text'],
1905             'rerank_score': result['rerank_score'],
1906             'vector_score': result['score']
1907         })
1908
1909     # Combine results
1910     return {
1911         'text': response['text'],
1912         'sources': sources,
1913         'metadata': {
1914             'llm': response['metadata'],
1915             'search_results': len(search_results)
1916         }
1917     }
1918
1919 def create_rag_processor(search_pipeline, llm_model_path: str,
1920                         context_size: int = 2048, max_tokens: int = 512) -> RAGProcessor:
1921     """
1922     Create a RAG processor.
1923
1924     Args:
1925         search_pipeline: Search pipeline
1926         llm_model_path: Path to the LLM model
1927         context_size: Context size for the model
1928         max_tokens: Maximum number of tokens to generate
1929
1930     Returns:
1931         RAGProcessor instance
1932     """

```

```

1928     RAG processor
1929     """
1930     # Create LLM processor
1931     llm_processor = LLMProcessor(llm_model_path, context_size, max_tokens)
1932
1933     # Create RAG processor
1934     rag_processor = RAGProcessor(search_pipeline, llm_processor)
1935
1936     return rag_processor
1937
1938 if __name__ == "__main__":
1939     # Example usage
1940     import sys
1941     from pathlib import Path
1942
1943     # Add the project root to the Python path
1944     sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
1945
1946     from app.config.settings import (
1947         VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
1948         EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH, LLM_MODEL_PATH
1949     )
1950     from app.utils.search import create_search_pipeline
1951
1952     # Create search pipeline
1953     search_pipeline = create_search_pipeline(
1954         VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
1955         EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH
1956     )
1957
1958     # Create RAG processor
1959     rag_processor = create_rag_processor(search_pipeline, LLM_MODEL_PATH)
1960
1961     # Process query
1962     query = "What is retrieval-augmented generation?"
1963     response = rag_processor.process_query(query)
1964
1965     # Print response
1966     print(f"Query: {query}")
1967     print(f"Response: {response['text']}")
1968     print(f"Sources: {len(response['sources'])}")
1969     for i, source in enumerate(response['sources']):
1970         print(f"Source {i+1}: {source['filename']} (Score: {source['rerank_score']:.4f})")

```

**Professor's Hint:** Prompt engineering is critical for effective RAG systems. The prompt should guide the LLM to focus on the provided context and avoid hallucination. Setting a lower temperature (e.g., 0.2) helps produce more factual, deterministic responses based on the context.

### 3. Create a complete RAG application:

```
touch app/rag_app.py
```

4. Add the following to app/rag\_app.py:

```
import os
import sys
import argparse
import logging
from pathlib import Path

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

# Add the project root to the Python path
sys.path.insert(0, str(Path(__file__).resolve().parent.parent))

from app.config.settings import (
    VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
    EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH, LLM_MODEL_PATH
)

from app.utils.search import create_search_pipeline
from app.utils.llm import create_rag_processor

class RAGApplication:
    def __init__(self):
        """Initialize the RAG application."""
        logger.info("Initializing RAG application")

        # Create search pipeline
        self.search_pipeline = create_search_pipeline(
            VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
            EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH
        )

        # Create RAG processor
        self.rag_processor = create_rag_processor(self.search_pipeline, LLM_MODEL_PATH)

        logger.info("RAG application initialized")

    def process_query(self, query: str):
        """
        Process a query.

        Args:
            query: User query

        Returns:
            RAG response
```

```

2022         """
2023         return self.rag_processor.process_query(query)
2024
2025 def interactive_mode(app):
2026     """Run the application in interactive mode."""
2027     print("RAG Application - Interactive Mode")
2028     print("Type 'exit' or 'quit' to end the session")
2029     print("-----")
2030
2031     while True:
2032         query = input("\nEnter your question: ")
2033
2034         if query.lower() in ['exit', 'quit']:
2035             print("Exiting...")
2036             break
2037
2038         try:
2039             response = app.process_query(query)
2040
2041             print("\nAnswer:")
2042             print(response['text'])
2043
2044             print("\nSources:")
2045             for i, source in enumerate(response['sources']):
2046                 print(f"{i+1}. {source['filename']} (Score: {source['rerank_score']:.4f})")
2047                 print(f"    Excerpt: {source['chunk_text'][:100]}...")
2048
2049             print("\nMetadata:")
2050             print(f"Total tokens: {response['metadata']['llm']['tokens_used']}")
2051             print(f"Search results: {response['metadata']['search_results']}")
2052
2053         except Exception as e:
2054             logger.error(f"Error processing query: {str(e)}")
2055             print(f"Error: {str(e)}")
2056
2057 if __name__ == "__main__":
2058     # Parse arguments
2059     parser = argparse.ArgumentParser(description='Run the RAG application')
2060     parser.add_argument('--interactive', action='store_true',
2061                         help='Run in interactive mode')
2062     parser.add_argument('--query', type=str,
2063                         help='Query to process')
2064     args = parser.parse_args()
2065
2066     # Initialize application
2067     app = RAGApplication()
2068
2069     if args.interactive:
2070         interactive_mode(app)
2071     elif args.query:

```

```

response = app.process_query(args.query)
print(response['text'])
else:
    parser.print_help()

```

#### Checkpoint Questions:

1. How does the prompt structure influence the quality of LLM responses in a RAG system?
2. What LLM parameters (temperature, top\_p, etc.) are most important for RAG applications and why?
3. How do we handle the case where no relevant documents are found in the search phase?
4. What are the ethical considerations when an LLM generates answers that contradict the retrieved context?
5. How would you modify the system to support providing citations or references in the generated responses?
6. What techniques could you implement to reduce hallucinations in the generated answers?
7. How does the choice of context window size affect the tradeoff between comprehensive context and response quality?
8. **Additional Challenge:** Implement a document citation mechanism that links specific parts of the response to the source documents, allowing users to verify information.

#### Lab 5.2: HTML Templates and Static Files.

##### Learning Objectives:

- Create responsive HTML templates using Bootstrap
- Implement client-side functionality with JavaScript
- Design an intuitive user interface for RAG interactions

##### Exercises:

1. Create the base template:

```

mkdir -p flask-app/templates
touch flask-app/templates/base.html

```

2. Add the following to flask-app/templates/base.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{% block title %}Local RAG System{% endblock %}</title>

  <!-- Bootstrap CSS -->
  <link
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
    rel="stylesheet">

  <!-- Font Awesome -->

```

```

2116 <link rel="stylesheet"
2117 href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">
2118
2119 <!-- Custom CSS -->
2120 <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
2121
2122 {% block extra_css %}{% endblock %}
2123 </head>
2124 <body>
2125 <!-- Navigation -->
2126 <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
2127     <div class="container">
2128         <a class="navbar-brand" href="{{ url_for('index') }}">Local RAG System</a>
2129         <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
2130 data-bs-target="#navbarNav">
2131             <span class="navbar-toggler-icon"></span>
2132         </button>
2133         <div class="collapse navbar-collapse" id="navbarNav">
2134             <ul class="navbar-nav">
2135                 <li class="nav-item">
2136                     <a class="nav-link {% if request.path == url_for('index') %}active{%
2137 endif %}" href="{{ url_for('index') }}">Home</a>
2138                 </li>
2139                 <li class="nav-item">
2140                     <a class="nav-link {% if request.path == url_for('upload') %}active{%
2141 endif %}" href="{{ url_for('upload') }}">Upload</a>
2142                 </li>
2143                 <li class="nav-item">
2144                     <a class="nav-link {% if request.path == url_for('documents')
2145 %}active{% endif %}" href="{{ url_for('documents') }}">Documents</a>
2146                 </li>
2147                 <li class="nav-item">
2148                     <a class="nav-link {% if request.path == url_for('ask') %}active{%
2149 endif %}" href="{{ url_for('ask') }}">Ask Questions</a>
2150                 </li>
2151             </ul>
2152
2153             <!-- System Status -->
2154             <div class="ms-auto">
2155                 <span class="navbar-text" id="system-status">
2156                     <i class="fas fa-circle-notch fa-spin"></i> Checking system status...
2157                 </span>
2158             </div>
2159         </div>
2160     </div>
2161 </nav>
2162
2163 <!-- Main Content -->
2164 <div class="container my-4">
2165     <!-- Flash Messages -->

```

```

2163     {% with messages = get_flashed_messages(with_categories=true) %}
2164     {% if messages %}
2165         {% for category, message in messages %}
2166             <div class="alert alert-{{ category if category != 'message' else 'info'
2167 }} alert-dismissible fade show">
2168                 {{ message }}
2169                 <button type="button" class="btn-close"
2170 data-bs-dismiss="alert"></button>
2171             </div>
2172         {% endfor %}
2173     {% endif %}
2174     {% endwith %}
2175
2176     <!-- Page Content -->
2177     {% block content %}{% endblock %}
2178 </div>
2179
2180 <!-- Footer -->
2181 <footer class="bg-light py-3 mt-5">
2182     <div class="container text-center">
2183         <p class="mb-0">Local RAG System &copy; 2025</p>
2184     </div>
2185 </footer>
2186
2187 <!-- Bootstrap JS Bundle -->
2188 <script
2189 src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
2190
2191 <!-- System Status Check -->
2192 <script>
2193     document.addEventListener('DOMContentLoaded', function() {
2194         // Check system status
2195         fetch('/api/health')
2196         .then(response => response.json())
2197         .then(data => {
2198             const statusEl = document.getElementById('system-status');
2199             if (data.mlflow) {
2200                 statusEl.innerHTML = '<i class="fas fa-check-circle text-success"></i>
2201 System Ready';
2202                 statusEl.classList.add('text-success');
2203             } else {
2204                 statusEl.innerHTML = '<i class="fas fa-exclamation-circle
2205 text-warning"></i> MLflow Offline';
2206                 statusEl.classList.add('text-warning');
2207             }
2208         })
2209         .catch(error => {
2210             console.error('Error checking status:', error);
2211             const statusEl = document.getElementById('system-status');

```

```

2210         statusEl.innerHTML = '<i class="fas fa-times-circle text-danger"></i>
2211     System Error';
2212         statusEl.classList.add('text-danger');
2213     });
2214 }
2215 </script>
2216 <!-- Extra JavaScript -->
2217 {% block extra_js %}{% endblock %}
2218 </body>
2219 </html>
2220

```

### 3. Create the main CSS file:

```

2223 mkdir -p flask-app/static/css
2224 touch flask-app/static/css/style.css
2225

```

### 4. Add the following to flask-app/static/css/style.css:

```

2227 /* Custom styles for the RAG system */
2228
2229 /* Main container min-height for short pages */
2230 body {
2231     display: flex;
2232     flex-direction: column;
2233     min-height: 100vh;
2234 }
2235
2236 .container {
2237     flex: 1;
2238 }
2239
2240 footer {
2241     margin-top: auto;
2242 }
2243
2244 /* Card hover effect */
2245 .card-hover:hover {
2246     box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
2247     transform: translateY(-2px);
2248     transition: all 0.3s ease;
2249 }
2250
2251 /* Document list */
2252 .document-list .document-item {
2253     border-left: 4px solid #007bff;
2254     margin-bottom: 10px;
2255 }
2256
2257 /* Source highlighting */

```



```

2257 .source-highlight {
2258     background-color: rgba(255, 243, 205, 0.5);
2259     border-radius: 3px;
2260     padding: 2px 4px;
2261 }
2262
2263 /* Answer section */
2264 .answer-section {
2265     background-color: #f8f9fa;
2266     border-radius: 5px;
2267     padding: 20px;
2268     margin-top: 20px;
2269 }
2270 .answer-text {
2271     font-size: 1.1rem;
2272     line-height: 1.6;
2273 }
2274
2275 .source-section {
2276     margin-top: 20px;
2277     border-top: 1px solid #dee2e6;
2278     padding-top: 15px;
2279 }
2280 .source-item {
2281     padding: 10px;
2282     margin-bottom: 10px;
2283     border-radius: 4px;
2284     background-color: #fff;
2285     border: 1px solid #e9ecef;
2286 }
2287 .source-text {
2288     max-height: 150px;
2289     overflow-y: auto;
2290 }
2291
2292 /* Loading spinner */
2293 .spinner-container {
2294     display: flex;
2295     justify-content: center;
2296     align-items: center;
2297     min-height: 200px;
2298 }
2299
2300 /* Citation tooltips */
2301 .citation {
2302     cursor: pointer;
2303     color: #007bff;

```

```

font-weight: bold;
font-size: 0.8rem;
vertical-align: super;
}

```

## 5. Create the index page:

```
touch flask-app/templates/index.html
```

## 6. Add the following to flask-app/templates/index.html:

```

{% extends 'base.html' %}

{% block title %}Home - Local RAG System{% endblock %}

{% block content %}
<div class="row">
  <div class="col-md-8 offset-md-2 text-center">
    <h1 class="display-4 mb-4">Local RAG System</h1>
    <p class="lead mb-5">
      A complete Retrieval-Augmented Generation system running locally on your MacMini.
      Upload PDF documents, ask questions, and get AI-generated answers based on your
      documents.
    </p>

    <div class="row mb-5">
      <div class="col-md-4">
        <div class="card card-hover h-100">
          <div class="card-body text-center">
            <i class="fas fa-file-upload fa-3x text-primary mb-3"></i>
            <h5 class="card-title">Upload Documents</h5>
            <p class="card-text">Add your PDF files to the knowledge base.</p>
            <a href="{{ url_for('upload') }}" class="btn
btn-outline-primary">Upload Files</a>
          </div>
        </div>
      </div>

      <div class="col-md-4">
        <div class="card card-hover h-100">
          <div class="card-body text-center">
            <i class="fas fa-list fa-3x text-primary mb-3"></i>
            <h5 class="card-title">Manage Documents</h5>
            <p class="card-text">View and manage your uploaded documents.</p>
            <a href="{{ url_for('documents') }}" class="btn
btn-outline-primary">View Documents</a>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

```

2351         <div class="col-md-4">
2352             <div class="card card-hover h-100">
2353                 <div class="card-body text-center">
2354                     <i class="fas fa-question-circle fa-3x text-primary mb-3"></i>
2355                     <h5 class="card-title">Ask Questions</h5>
2356                     <p class="card-text">Get answers based on your documents.</p>
2357                     <a href="{{ url_for('ask') }}" class="btn btn-outline-primary">Ask
Now</a>
2358                 </div>
2359             </div>
2360         </div>
2361     </div>
2362
2363     <div class="card bg-light">
2364         <div class="card-body">
2365             <h4>How It Works</h4>
2366             <hr>
2367             <div class="row">
2368                 <div class="col-md-3 text-center">
2369                     <i class="fas fa-file-pdf fa-2x text-danger"></i>
2370                     <h6 class="mt-2">1. Upload</h6>
2371                     <p class="small">Add PDF documents to build your knowledge base</p>
2372                 </div>
2373                 <div class="col-md-3 text-center">
2374                     <i class="fas fa-project-diagram fa-2x text-primary"></i>
2375                     <h6 class="mt-2">2. Process</h6>
2376                     <p class="small">Documents are chunked and indexed</p>
2377                 </div>
2378                 <div class="col-md-3 text-center">
2379                     <i class="fas fa-search fa-2x text-success"></i>
2380                     <h6 class="mt-2">3. Search</h6>
2381                     <p class="small">Your question finds relevant content</p>
2382                 </div>
2383                 <div class="col-md-3 text-center">
2384                     <i class="fas fa-robot fa-2x text-info"></i>
2385                     <h6 class="mt-2">4. Answer</h6>
2386                     <p class="small">AI generates an answer from the sources</p>
2387                 </div>
2388             </div>
2389         </div>
2390     </div>
2391 {% endblock %}

```

## 7. Create the upload page:

```
touch flask-app/templates/upload.html
```

## 8. Add the following to flask-app/templates/upload.html:

```

2398 {% extends 'base.html' %}
2399
2400 {% block title %}Upload Documents - Local RAG System{% endblock %}
2401
2402 {% block content %}
2403 <div class="row">
2404     <div class="col-md-8 offset-md-2">
2405         <h1 class="mb-4">Upload Documents</h1>
2406
2407         <div class="card mb-4">
2408             <div class="card-body">
2409                 <h5 class="card-title">Add PDF Files</h5>
2410                 <p class="card-text">Upload PDF documents to be processed and added to the
2411 knowledge base.</p>
2412
2413                 <form method="POST" enctype="multipart/form-data" class="mt-4">
2414                     <div class="mb-3">
2415                         <label for="file" class="form-label">Select PDF File</label>
2416                         <input type="file" class="form-control" id="file" name="file"
2417 accept=".pdf" required>
2418                         <div class="form-text">Maximum file size: 16MB</div>
2419                     </div>
2420
2421                     <div class="mb-3 form-check">
2422                         <input type="checkbox" class="form-check-input" id="rebuild"
2423 name="rebuild">
2424                         <label class="form-check-label" for="rebuild">
2425                             Rebuild index after upload (slower but ensures immediate
2426 availability)
2427                         </label>
2428                     </div>
2429
2430                     <div class="d-grid">
2431                         <button type="submit" class="btn btn-primary">
2432                             <i class="fas fa-upload me-2"></i> Upload Document
2433                         </button>
2434                     </div>
2435                 </form>
2436             </div>
2437         </div>
2438
2439         <div class="card">
2440             <div class="card-body">
2441                 <h5 class="card-title">Batch Upload</h5>
2442                 <p class="card-text">
2443                     For uploading multiple files or large documents, you can directly copy
2444                     files to the documents directory:
2445                 </p>
2446                 <div class="bg-light p-3 rounded">

```

```

2445         <code id="pdf-folder-path">{{ config['UPLOAD_FOLDER'] }}</code>
2446         <button class="btn btn-sm btn-outline-secondary ms-2"
2447         onclick="copyToClipboard('pdf-folder-path')">
2448             <i class="fas fa-copy"></i>
2449         </button>
2450     </div>
2451     <p class="mt-2 mb-0 small text-muted">
2452         After copying files, you'll need to run the processing pipeline to index
2453         the documents.
2454     </p>
2455 </div>
2456 </div>
2457 </div>
2458 {% block extra_js %}
2459 <script>
2460     function copyToClipboard(elementId) {
2461         const element = document.getElementById(elementId);
2462         const text = element.textContent;
2463
2464         navigator.clipboard.writeText(text).then(function() {
2465             // Show a temporary success message
2466             const button = element.nextElementSibling;
2467             const originalIcon = button.innerHTML;
2468             button.innerHTML = '<i class="fas fa-check"></i>';
2469             button.classList.add('btn-success');
2470             button.classList.remove('btn-outline-secondary');
2471
2472             // Restore original after 2 seconds
2473             setTimeout(function() {
2474                 button.innerHTML = originalIcon;
2475                 button.classList.remove('btn-success');
2476                 button.classList.add('btn-outline-secondary');
2477             }, 2000);
2478         });
2479     </script>
2480 {% endblock %}
2481 {% endblock %}

```

## 9. Create the documents page:

```
touch flask-app/templates/documents.html
```

## 10. Add the following to flask-app/templates/documents.html:

```

2488 {% extends 'base.html' %}
2489
2490 {% block title %}Document Management - Local RAG System{% endblock %}
2491

```

```

2492 {% block content %}
2493 <div class="row">
2494   <div class="col-md-10 offset-md-1">
2495     <div class="d-flex justify-content-between align-items-center mb-4">
2496       <h1>Document Management</h1>
2497       <div>
2498         <a href="{% url_for('upload') %}" class="btn btn-primary">
2499           <i class="fas fa-upload me-2"></i> Upload New
2500         </a>
2501         <button id="reindex-btn" class="btn btn-outline-secondary ms-2">
2502           <i class="fas fa-sync me-2"></i> Reindex All
2503         </button>
2504       </div>
2505     </div>
2506
2507     {% if documents %}
2508       <div class="card">
2509         <div class="card-header bg-primary text-white">
2510           <div class="row">
2511             <div class="col-md-6">Filename</div>
2512             <div class="col-md-2">Size</div>
2513             <div class="col-md-3">Last Modified</div>
2514             <div class="col-md-1">Actions</div>
2515           </div>
2516         <div class="card-body p-0 document-list">
2517           {% for doc in documents %}
2518             <div class="document-item p-3 {% if loop.index % 2 == 0 %}bg-light{%
2519 endif %}">
2520               <div class="row align-items-center">
2521                 <div class="col-md-6">
2522                   <i class="fas fa-file-pdf text-danger me-2"></i>
2523                   {{ doc.filename }}
2524                 </div>
2525                 <div class="col-md-2">
2526                   {{ (doc.size / 1024)|round(1) }} KB
2527                 </div>
2528                 <div class="col-md-3">
2529                   {{ doc.modified|timestamp_to_date }}
2530                 </div>
2531                 <div class="col-md-1 text-end">
2532                   <div class="dropdown">
2533                     <button class="btn btn-sm btn-outline-secondary
2534 dropdown-toggle" type="button" data-bs-toggle="dropdown">
2535                       <i class="fas fa-ellipsis-v"></i>
2536                     </button>
2537                     <ul class="dropdown-menu dropdown-menu-end">
2538                       <li>

```

```

2539         <a class="dropdown-item" href="#"
2540         onclick="event.preventDefault(); deleteDocument('{{ doc.filename }}')">
2541             <i class="fas fa-trash text-danger
2542         me-2"></i> Delete
2543         </a>
2544     </li>
2545     <li>
2546         <a class="dropdown-item" href="#"
2547         onclick="event.preventDefault(); reindexDocument('{{ doc.filename }}')">
2548             <i class="fas fa-sync text-primary
2549         me-2"></i> Reindex
2550         </a>
2551     </li>
2552 </ul>
2553 </div>
2554 </div>
2555 </div>
2556     {% endfor %}
2557 </div>
2558 <div class="card-footer">
2559     <small class="text-muted">{{ documents|length }} document(s)</small>
2560 </div>
2561 {% else %}
2562 <div class="alert alert-info">
2563     <i class="fas fa-info-circle me-2"></i> No documents uploaded yet.
2564     <a href="{{ url_for('upload') }}" class="alert-link">Upload your first
2565     document</a>.
2566 </div>
2567 {% endif %}
2568 </div>
2569
2570 <!-- Delete Confirmation Modal -->
2571 <div class="modal fade" id="deleteModal" tabindex="-1">
2572     <div class="modal-dialog">
2573         <div class="modal-content">
2574             <div class="modal-header">
2575                 <h5 class="modal-title">Confirm Deletion</h5>
2576                 <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
2577             </div>
2578             <div class="modal-body">
2579                 <p>Are you sure you want to delete <strong id="deleteFileName"></strong>?</p>
2580                 <p class="text-danger mb-0">This action cannot be undone.</p>
2581             </div>
2582             <div class="modal-footer">
2583                 <button type="button" class="btn btn-secondary"
2584                 data-bs-dismiss="modal">Cancel</button>
2585                 <button type="button" class="btn btn-danger" id="confirmDelete">Delete</button>

```

```

2586         </div>
2587     </div>
2588 </div>
2589 </div>
2590
2591 <!-- Reindex Modal -->
2592 <div class="modal fade" id="reindexModal" tabindex="-1">
2593     <div class="modal-dialog">
2594         <div class="modal-content">
2595             <div class="modal-header">
2596                 <h5 class="modal-title">Reindex Documents</h5>
2597                 <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
2598             </div>
2599             <div class="modal-body">
2600                 <p>This will reprocess all documents and update the search index.</p>
2601                 <p>Depending on the number of documents, this may take some time.</p>
2602             </div>
2603             <div class="modal-footer">
2604                 <button type="button" class="btn btn-secondary"
2605                 data-bs-dismiss="modal">Cancel</button>
2606                 <button type="button" class="btn btn-primary"
2607                 id="confirmReindex">Reindex</button>
2608             </div>
2609         </div>
2610     </div>
2611 </div>
2612
2613 {% block extra_js %}
2614 <script>
2615     // Delete document
2616     function deleteDocument(filename) {
2617         document.getElementById('deleteFileName').textContent = filename;
2618         const deleteModal = new bootstrap.Modal(document.getElementById('deleteModal'));
2619
2620         document.getElementById('confirmDelete').onclick = function() {
2621             // Send delete request
2622             fetch('/api/documents/delete', {
2623                 method: 'POST',
2624                 headers: {
2625                     'Content-Type': 'application/json',
2626                 },
2627                 body: JSON.stringify({
2628                     filename: filename
2629                 }),
2630             })
2631             .then(response => response.json())
2632             .then(data => {
2633                 if (data.success) {
2634                     // Reload the page
2635                     window.location.reload();
2636                 }
2637             });
2638         };
2639     }
2640 </script>
2641 %}

```



```

2633         } else {
2634             alert('Error: ' + data.error);
2635         }
2636     })
2637     .catch(error => {
2638         console.error('Error:', error);
2639         alert('An error occurred while deleting the document.');
```

```

2640     });
2641     deleteModal.hide();
2642 };
2643
2644 deleteModal.show();
2645 }
2646
2647 // Reindex single document
2648 function reindexDocument(filename) {
2649     // Send reindex request
2650     fetch('/api/documents/reindex', {
2651         method: 'POST',
2652         headers: {
2653             'Content-Type': 'application/json',
2654         },
2655         body: JSON.stringify({
2656             filename: filename
2657         }),
2658     })
2659     .then(response => response.json())
2660     .then(data => {
2661         if (data.success) {
2662             alert('Reindexing started. This may take a few minutes.');
```

```

2663         } else {
2664             alert('Error: ' + data.error);
2665         }
2666     })
2667     .catch(error => {
2668         console.error('Error:', error);
2669         alert('An error occurred while reindexing the document.');
```

```

2670     });
2671 }
2672
2673 // Reindex all documents
2674 document.getElementById('reindex-btn').addEventListener('click', function() {
2675     const reindexModal = new bootstrap.Modal(document.getElementById('reindexModal'));
2676
2677     document.getElementById('confirmReindex').onclick = function() {
2678         // Send reindex all request
2679         fetch('/api/documents/reindex-all', {
2680             method: 'POST',
2681             headers: {
```

```

2680         'Content-Type': 'application/json',
2681     },
2682 })
2683 .then(response => response.json())
2684 .then(data => {
2685     if (data.success) {
2686         alert('Reindexing started. This may take a few minutes.');
```

2703 11. Create the ask page:

```
2704 touch flask-app/templates/ask.html
```

2707 12. Add the following to flask-app/templates/ask.html:

```

2708 {% extends 'base.html' %}
2709
2710 {% block title %}Ask Questions - Local RAG System{% endblock %}
2711
2712 {% block content %}
2713 <div class="row">
2714     <div class="col-md-10 offset-md-1">
2715         <h1 class="mb-4">Ask Questions</h1>
2716
2717         <div class="card mb-4">
2718             <div class="card-body">
2719                 <form id="question-form">
2720                     <div class="mb-3">
2721                         <label for="question" class="form-label">Your Question</label>
2722                         <textarea class="form-control" id="question" rows="3"
2723                             placeholder="Enter your question here..." required></textarea>
2724                     </div>
2725                     <div class="d-grid">
2726                         <button type="submit" class="btn btn-primary" id="submit-btn">
```

```

2727         <i class="fas fa-search me-2"></i> Ask Question
2728     </button>
2729 </div>
2730 </form>
2731 </div>
2732 </div>
2733 <!-- Answer section (initially hidden) -->
2734 <div id="answer-container" class="mb-4" style="display: none;">
2735     <div class="card">
2736         <div class="card-header bg-primary text-white">
2737             <h5 class="mb-0">Answer</h5>
2738         </div>
2739         <div class="card-body">
2740             <div class="answer-text" id="answer-text">
2741                 <!-- Answer will be inserted here -->
2742             </div>
2743         </div>
2744     </div>
2745 </div>
2746 <!-- Sources section (initially hidden) -->
2747 <div id="sources-container" style="display: none;">
2748     <h4>Sources</h4>
2749     <div id="sources-list">
2750         <!-- Sources will be inserted here -->
2751     </div>
2752 </div>
2753 <!-- Loading spinner (initially hidden) -->
2754 <div id="loading-spinner" class="spinner-container" style="display: none;">
2755     <div class="text-center">
2756         <div class="spinner-border text-primary" role="status">
2757             <span class="visually-hidden">Loading...</span>
2758         </div>
2759         <p class="mt-2">Processing your question...</p>
2760     </div>
2761 </div>
2762 <!-- Error message (initially hidden) -->
2763 <div id="error-message" class="alert alert-danger" style="display: none;">
2764     <!-- Error will be inserted here -->
2765 </div>
2766 </div>
2767 </div>
2768
2769 {% block extra_js %}
2770 <script>
2771     document.addEventListener('DOMContentLoaded', function() {
2772         const questionForm = document.getElementById('question-form');
2773

```

```

2774     const submitBtn = document.getElementById('submit-btn');
2775     const loadingSpinner = document.getElementById('loading-spinner');
2776     const answerContainer = document.getElementById('answer-container');
2777     const answerText = document.getElementById('answer-text');
2778     const sourcesContainer = document.getElementById('sources-container');
2779     const sourcesList = document.getElementById('sources-list');
2780     const errorMessage = document.getElementById('error-message');
2781
2782     questionForm.addEventListener('submit', function(e) {
2783         e.preventDefault();
2784
2785         // Get the question
2786         const question = document.getElementById('question').value.trim();
2787         if (!question) return;
2788
2789         // Show loading spinner
2790         loadingSpinner.style.display = 'flex';
2791         answerContainer.style.display = 'none';
2792         sourcesContainer.style.display = 'none';
2793         errorMessage.style.display = 'none';
2794         submitBtn.disabled = true;
2795
2796         // Send the question to the API
2797         fetch('/api/ask', {
2798             method: 'POST',
2799             headers: {
2800                 'Content-Type': 'application/json',
2801             },
2802             body: JSON.stringify({
2803                 question: question
2804             }),
2805         })
2806         .then(response => response.json())
2807         .then(data => {
2808             // Hide loading spinner
2809             loadingSpinner.style.display = 'none';
2810             submitBtn.disabled = false;
2811
2812             if (data.error) {
2813                 // Show error message
2814                 errorMessage.textContent = data.error;
2815                 errorMessage.style.display = 'block';
2816                 return;
2817             }
2818
2819             // Show answer
2820             answerText.innerHTML = data.text;
2821             answerContainer.style.display = 'block';
2822
2823             // Show sources if available

```

```

2821         if (data.sources && data.sources.length > 0) {
2822             // Clear previous sources
2823             sourcesList.innerHTML = '';
2824
2825             // Add each source
2826             data.sources.forEach((source, index) => {
2827                 const sourceElement = document.createElement('div');
2828                 sourceElement.className = 'source-item';
2829                 sourceElement.innerHTML = `
2830 mb-2">
2831                 <h6 class="mb-0">
2832                     <i class="fas fa-file-pdf text-danger me-2"></i>
2833                     ${source.filename}
2834                 </h6>
2835                 <span class="badge bg-primary">Score:
2836 ${source.rerank_score.toFixed(2)}</span>
2837                 </div>
2838                 <div class="source-text">
2839                     <p class="mb-0">${source.chunk_text}</p>
2840                 </div>
2841                 `;
2842                 sourcesList.appendChild(sourceElement);
2843             });
2844             sourcesContainer.style.display = 'block';
2845         }
2846     })
2847     .catch(error => {
2848         console.error('Error:', error);
2849         loadingSpinner.style.display = 'none';
2850         submitBtn.disabled = false;
2851         errorMessage.textContent = 'An error occurred while processing your question.
Please try again.';
2852         errorMessage.style.display = 'block';
2853     });
2854 });
2855 });
2856 </script>
2857 {% endblock %}
2858 {% endblock %}

```

**Professor's Hint:** Always implement robust error handling in your web interfaces, especially for operations that involve AI processing which can sometimes be unpredictable. The spinner provides important feedback to users during potentially lengthy operations, while the clear display of source documents helps users understand and trust the generated answers.

**Lab 5.3: Complete the Flask Application.**

*Learning Objectives:*

- Implement the remaining API endpoints for the Flask application
- Add custom filters and utility functions
- Test the complete web interface

*Exercises:*

1. Add the following additional API endpoints to `app.py`:

```
# Add these imports at the top
import datetime
import shutil

# Register custom Jinja2 filters
@app.template_filter('timestamp_to_date')
def timestamp_to_date(timestamp):
    """Convert a timestamp to a formatted date string."""
    dt = datetime.datetime.fromtimestamp(timestamp)
    return dt.strftime('%Y-%m-%d %H:%M')

@app.route('/api/documents/delete', methods=['POST'])
def api_delete_document():
    """API endpoint for deleting a document."""
    data = request.get_json()

    if not data or 'filename' not in data:
        return jsonify({'success': False, 'error': 'Missing filename parameter'}), 400

    filename = secure_filename(data['filename'])
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)

    if not os.path.exists(file_path):
        return jsonify({'success': False, 'error': 'File not found'}), 404

    try:
        # Delete the file
        os.remove(file_path)

        # Note: In a production system, we would also want to remove the document from the
        # vector database

        return jsonify({'success': True})
    except Exception as e:
        logger.error(f"Error deleting document: {str(e)}")
        return jsonify({'success': False, 'error': str(e)}), 500

@app.route('/api/documents/reindex', methods=['POST'])
def api_reindex_document():
    """API endpoint for reindexing a document."""
    data = request.get_json()

    if not data or 'filename' not in data:
```

```

2915         return jsonify({'success': False, 'error': 'Missing filename parameter'}), 400
2916
2917     filename = secure_filename(data['filename'])
2918     file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
2919
2920     if not os.path.exists(file_path):
2921         return jsonify({'success': False, 'error': 'File not found'}), 404
2922
2923     try:
2924         # Import pipeline trigger module
2925         from flask_app.utils.pipeline_trigger import run_pipeline_async
2926
2927         # Create a temporary directory with just this file
2928         temp_dir = os.path.join(app.config['UPLOAD_FOLDER'], '_temp_reindex')
2929         os.makedirs(temp_dir, exist_ok=True)
2930
2931         # Copy the file to the temp directory
2932         shutil.copy(file_path, os.path.join(temp_dir, filename))
2933
2934         # Run the pipeline on the temp directory
2935         result = run_pipeline_async(temp_dir, rebuild=True)
2936
2937         return jsonify({'success': True})
2938     except Exception as e:
2939         logger.error(f"Error reindexing document: {str(e)}")
2940         return jsonify({'success': False, 'error': str(e)}), 500
2941
2942 @app.route('/api/documents/reindex-all', methods=['POST'])
2943 def api_reindex_all():
2944     """API endpoint for reindexing all documents."""
2945     try:
2946         # Import pipeline trigger module
2947         from flask_app.utils.pipeline_trigger import run_pipeline_async
2948
2949         # Run the pipeline on the upload folder
2950         result = run_pipeline_async(app.config['UPLOAD_FOLDER'], rebuild=True)
2951
2952         return jsonify({'success': True})
2953     except Exception as e:
2954         logger.error(f"Error reindexing all documents: {str(e)}")
2955         return jsonify({'success': False, 'error': str(e)}), 500

```

## 2. Create a modified version of the upload route to handle processing:

```

2956 @app.route('/upload', methods=['GET', 'POST'])
2957 def upload():
2958     """Handle file uploads."""
2959     if request.method == 'POST':
2960         # Check if the post request has the file part
2961         if 'file' not in request.files:

```

```

2962         flash('No file part', 'error')
2963         return redirect(request.url)
2964
2965     file = request.files['file']
2966
2967     # If user does not select file, browser also
2968     # submit an empty part without filename
2969     if file.filename == '':
2970         flash('No selected file', 'error')
2971         return redirect(request.url)
2972
2973     if file and allowed_file(file.filename):
2974         filename = secure_filename(file.filename)
2975         file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
2976         file.save(file_path)
2977
2978         # Check if we should rebuild the index
2979         rebuild = request.form.get('rebuild', 'off') == 'on'
2980
2981         if rebuild:
2982             try:
2983                 # Import pipeline trigger module
2984                 from flask_app.utils.pipeline_trigger import run_pipeline_async
2985
2986                 # Run the pipeline
2987                 run_pipeline_async(app.config['UPLOAD_FOLDER'], rebuild=rebuild)
2988
2989                 flash(f'File {filename} uploaded successfully and indexing started!',
2990                       'success')
2991             except Exception as e:
2992                 logger.error(f"Error running pipeline: {str(e)}")
2993                 flash(f'File uploaded but indexing failed: {str(e)}', 'warning')
2994             else:
2995                 flash(f'File {filename} uploaded successfully!', 'success')
2996
2997                 # Redirect to document list
2998                 return redirect(url_for('documents'))
2999         else:
3000             flash('File type not allowed', 'error')
3001             return redirect(request.url)
3002
3003     return render_template('upload.html')

```

### 3. Create a startup script to launch the entire system:

```

touch startup.sh
chmod +x startup.sh

```

### 4. Add the following to startup.sh:



```

3009 #!/bin/bash
3010
3011 # Exit on error
3012 set -e
3013
3014 # Configuration
3015 PROJECT_ROOT=$(pwd)
3016 PDF_DIR="${PROJECT_ROOT}/data/pdfs"
3017 MODELS_DIR="${PROJECT_ROOT}/models"
3018 VENV_DIR="${PROJECT_ROOT}/venv"
3019
3020 # Colors for output
3021 GREEN='\033[0;32m'
3022 YELLOW='\033[1;33m'
3023 RED='\033[0;31m'
3024 NC='\033[0m' # No Color
3025
3026 echo -e "${GREEN}Starting Local RAG System...${NC}"
3027
3028 # Check for virtual environment
3029 if [ ! -d "$VENV_DIR" ]; then
3030     echo -e "${YELLOW}Virtual environment not found. Creating...${NC}"
3031     python3 -m venv venv
3032 fi
3033
3034 # Activate virtual environment
3035 echo "Activating virtual environment..."
3036 source venv/bin/activate
3037
3038 # Check for requirements
3039 if [ ! -f "requirements.txt" ]; then
3040     echo -e "${RED}Error: requirements.txt not found.${NC}"
3041     exit 1
3042 fi
3043
3044 # Install requirements if needed
3045 pip -q install -r requirements.txt
3046
3047 # Check for models
3048 if [ ! -d "$MODELS_DIR/embedding" ] || [ ! -d "$MODELS_DIR/reranker" ] || [ ! -d
3049     "$MODELS_DIR/llm" ]; then
3050     echo -e "${YELLOW}Some models are missing. Please download them first.${NC}"
3051     echo "You can use the following commands:"
3052     echo "  python app/scripts/download_models.py"
3053     echo "  ./app/scripts/download_llm.sh"
3054 fi
3055
3056 # Create data directories if they don't exist
3057 mkdir -p "$PDF_DIR"

```

```

3056
3057 # Start Docker services
3058 echo "Starting Docker services..."
3059 docker-compose up -d
3060
3061 # Wait for services to be ready
3062 echo "Waiting for services to be ready..."
3063 sleep 5
3064
3065 # Check if MLflow is running
3066 echo "Checking MLflow service..."
3067 if curl -s http://localhost:5001/ping > /dev/null; then
3068     echo -e "${GREEN}MLflow service is running.${NC}"
3069 else
3070     echo -e "${RED}MLflow service is not running. Check Docker logs.${NC}"
3071     echo "You can run: docker-compose logs mlflow"
3072 fi
3073
3074 # Check if vector database is running
3075 echo "Checking vector database service..."
3076 if curl -s http://localhost:6333/health > /dev/null; then
3077     echo -e "${GREEN}Vector database service is running.${NC}"
3078 else
3079     echo -e "${RED}Vector database service is not running. Check Docker logs.${NC}"
3080     echo "You can run: docker-compose logs vector-db"
3081 fi
3082
3083 # Check if there are PDFs to process
3084 PDF_COUNT=$(find "$PDF_DIR" -name "*.pdf" | wc -l)
3085 if [ "$PDF_COUNT" -gt 0 ]; then
3086     echo -e "${GREEN}Found $PDF_COUNT PDF files.${NC}"
3087
3088     # Ask if user wants to process them
3089     read -p "Do you want to process them now? (y/n) " -n 1 -r
3090     echo
3091     if [[ $REPLY =~ ^[Yy]$ ]]; then
3092         echo "Running processing pipeline..."
3093         python app/pipeline.py
3094     else
3095         echo "Skipping processing."
3096     fi
3097 else
3098     echo "No PDF files found. You can upload them through the web interface."
3099 fi
3100
3101 # Deploy the model to MLflow
3102 echo "Deploying model to MLflow..."
3103 # First, check if model exists
3104 if mlflow models search -f "name = 'rag_model'" | grep -q "rag_model"; then
3105     echo "Model already exists, deploying latest version..."
3106

```

```

3103     python app/scripts/deploy_model.py &
3104 else
3105     echo "Model not found, logging new model..."
3106     python app/scripts/log_model.py
3107     python app/scripts/deploy_model.py &
3108 fi
3109 # Start Flask application
3110 echo -e "${GREEN}Starting Flask application...${NC}"
3111 cd flask-app
3112 FLASK_APP=app.py FLASK_ENV=development python app.py &
3113
3114 echo -e "${GREEN}All services started!${NC}"
3115 echo "You can access the web interface at: http://localhost:8000"
3116 echo -e "${YELLOW}Press Ctrl+C to stop all services${NC}"
3117
3118 # Wait for user interrupt
3119 trap "echo 'Stopping services...'; kill %1; docker-compose down; exit 0" INT
3120 wait

```

5. Create a Dockerfile for the Flask app:

```
touch flask-app/Dockerfile
```

6. Add the following to flask-app/Dockerfile:

```

3127 FROM python:3.10-slim
3128
3129 WORKDIR /app
3130
3131 COPY requirements.txt .
3132 RUN pip install --no-cache-dir -r requirements.txt
3133
3134 COPY . .
3135
3136 EXPOSE 8000
3137
3138 CMD ["python", "app.py"]

```

7. Update the Docker Compose file to include the Flask app:

```

3141 version: '3.8'
3142
3143 services:
3144   vector-db:
3145     image: qdrant/qdrant:latest
3146     ports:
3147       - "6333:6333"
3148       - "6334:6334"
3149     volumes:

```

```

3150     - ./data/vectors:/qdrant/storage
3151 environment:
3152     - QDRANT_ALLOW_CORS=true
3153 restart: unless-stopped
3154
3155 mlflow:
3156     image: ghcr.io/mlflow/mlflow:latest
3157     ports:
3158         - "5000:5000"
3159     volumes:
3160         - ./mlflow/artifacts:/mlflow/artifacts
3161         - ./mlflow/backend:/mlflow/backend
3162     environment:
3163         - MLFLOW_TRACKING_URI=sqlite:///mlflow/backend/mlflow.db
3164         - MLFLOW_DEFAULT_ARTIFACT_ROOT=/mlflow/artifacts
3165     command: mlflow server --backend-store-uri sqlite:///mlflow/backend/mlflow.db
3166             --default-artifact-root /mlflow/artifacts --host 0.0.0.0 --port 5000
3167     restart: unless-stopped
3168
3169 flask-app:
3170     build: ./flask-app
3171     ports:
3172         - "8000:8000"
3173     volumes:
3174         - ./flask-app:/app
3175         - ./data:/data
3176         - ./app:/rag_app
3177     environment:
3178         - FLASK_APP=app.py
3179         - FLASK_ENV=development
3180         - PYTHONPATH=/app:/rag_app
3181     depends_on:
3182         - vector-db
3183         - mlflow
3184     restart: unless-stopped

```

**Professor's Hint:** *The startup script is crucial for ensuring all components start in the correct order and with proper configuration. By handling dependency checks and proper error reporting, it makes the system much more robust and user-friendly.*

*Checkpoint Questions:*

1. How do Flask templates and static files work together to create a responsive user interface?
2. What are the advantages of using AJAX for form submissions in a web application?
3. How can we ensure the system continues running even when disconnected from the internet?
4. What UX considerations are specific to RAG interfaces compared to traditional search interfaces?
5. How would you implement progressive loading to improve perceived performance for long-running queries?
6. What accessibility considerations should be taken into account for a RAG interface?

7. How might you design the interface to help users formulate better queries and get more accurate results?
8. **Additional Challenge:** Enhance the web interface with a history of past questions and answers, allowing users to revisit their previous queries.

## Lab 6: Final Integration and Testing

### Lab 6.1: End-to-End System Integration.

#### Learning Objectives:

- Integrate all components into a cohesive system
- Test the complete RAG pipeline
- Troubleshoot common integration issues

#### Exercises:

1. Create an end-to-end integration test:

```
touch app/tests/test_end_to_end.py
```

2. Add the following to app/tests/test\_end\_to\_end.py:

```
import os
import sys
import time
import pytest
import requests
from pathlib import Path

# Add the project root to the Python path
sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))

from app.config.settings import PDF_UPLOAD_FOLDER, VECTOR_DB_HOST, VECTOR_DB_PORT,
    COLLECTION_NAME
from app.utils.vector_db import VectorDBClient
from app.clients.mlflow_client import create_mlflow_client

# Test configuration
FLASK_BASE_URL = "http://localhost:8000"
TEST_PDF_PATH = os.path.join(Path(__file__).resolve().parent, "data", "sample.pdf")
TEST_QUESTION = "What is machine learning?"

def check_service_availability():
    """Check if all services are available."""
    services = {
        "Flask Web App": f"{FLASK_BASE_URL}/api/health",
        "MLflow": f"http://localhost:5001/ping",
        "Vector DB": f"http://localhost:6333/health",
    }

    available = {}
```

```

3244     for name, url in services.items():
3245         try:
3246             response = requests.get(url, timeout=2)
3247             available[name] = response.status_code == 200
3248         except:
3249             available[name] = False
3250
3251     return available
3252
3253 @pytest.fixture(scope="module")
3254 def check_system():
3255     """Check if the entire system is ready for testing."""
3256     available = check_service_availability()
3257
3258     if not all(available.values()):
3259         unavailable = [name for name, status in available.items() if not status]
3260         pytest.skip(f"Some services are not available: {' '.join(unavailable)}")
3261
3262 def test_health_endpoints(check_system):
3263     """Test health endpoints of all services."""
3264     # Flask health
3265     response = requests.get(f"{FLASK_BASE_URL}/api/health")
3266     assert response.status_code == 200
3267     data = response.json()
3268     assert data["status"] == "ok"
3269
3270     # Check MLflow through Flask health endpoint
3271     assert data["mlflow"] == True
3272
3273 def test_vector_db_connection(check_system):
3274     """Test connection to vector database."""
3275     vector_db = VectorDBClient(VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, 384)
3276     assert vector_db.client is not None
3277
3278     # Check if collection exists
3279     collections = vector_db.client.get_collections().collections
3280     collection_names = [collection.name for collection in collections]
3281
3282     # If collection doesn't exist, this test is inconclusive
3283     if COLLECTION_NAME not in collection_names:
3284         pytest.skip(f"Collection {COLLECTION_NAME} does not exist yet")
3285
3286     # Check collection count
3287     count = vector_db.count_vectors()
3288     print(f"Vector count in collection: {count}")
3289
3290 def test_mlflow_client(check_system):
3291     """Test MLflow client."""
3292     mlflow_client = create_mlflow_client()
3293     assert mlflow_client.is_alive() == True

```

```

3291
3292 def test_document_list(check_system):
3293     """Test document list API."""
3294     response = requests.get(f"{FLASK_BASE_URL}/documents")
3295     assert response.status_code == 200
3296
3297 def test_ask_question(check_system):
3298     """Test asking a question."""
3299     response = requests.post(
3300         f"{FLASK_BASE_URL}/api/ask",
3301         json={"question": TEST_QUESTION}
3302     )
3303     assert response.status_code == 200
3304     data = response.json()
3305
3306     # Check response structure
3307     assert "text" in data
3308     assert len(data["text"]) > 0
3309
3310     # Sources might be empty if no relevant documents are found
3311     assert "sources" in data
3312
3313 def test_end_to_end_flow(check_system):
3314     """Test the complete end-to-end flow."""
3315     # This test is more of a recipe for manual testing
3316     print("\nEnd-to-end test steps:")
3317     print("1. Upload a PDF document through the web interface")
3318     print("2. Trigger indexing process")
3319     print("3. Wait for indexing to complete")
3320     print("4. Ask a question related to the document content")
3321     print("5. Verify that the answer references information from the document")
3322
3323     # Skip the actual test for automation
3324     pytest.skip("This test is a manual procedure")
3325
3326 if __name__ == "__main__":
3327     # Run tests
3328     pytest.main(["-xvs", __file__])
3329
3330
3331
3332
3333
3334
3335
3336
3337

```

3. Create a sample PDF for testing:

```
mkdir -p app/tests/data
```

4. Copy a small PDF to this directory for testing purposes.

5. Create a system status check script:

```
touch system_status.py
chmod +x system_status.py
```

6. Add the following to `system_status.py`:

```

3338
3339
3340 #!/usr/bin/env python3
3341 import os
3342 import sys
3343 import subprocess
3344 import requests
3345 import time
3346 import argparse
3347
3348 # ANSI colors
3349 class Colors:
3350     HEADER = '\033[95m'
3351     BLUE = '\033[94m'
3352     GREEN = '\033[92m'
3353     YELLOW = '\033[93m'
3354     RED = '\033[91m'
3355     ENDC = '\033[0m'
3356     BOLD = '\033[1m'
3357     UNDERLINE = '\033[4m'
3358
3359 def print_status(name, status, message=""):
3360     color = Colors.GREEN if status else Colors.RED
3361     status_text = "RUNNING" if status else "STOPPED"
3362     print(f"{name:20}: {color}{status_text}{Colors.ENDC} {message}")
3363
3364 def check_docker_service(service_name):
3365     try:
3366         output = subprocess.check_output(
3367             ["docker", "ps", "--filter", f"name={service_name}", "--format", "{{.Status}}"],
3368             stderr=subprocess.STDOUT,
3369             universal_newlines=True
3370         )
3371         return len(output.strip()) > 0, output.strip()
3372     except subprocess.CalledProcessError:
3373         return False, "Docker not running"
3374
3375 def check_http_endpoint(url, timeout=2):
3376     try:
3377         response = requests.get(url, timeout=timeout)
3378         return response.status_code == 200, f"Status code: {response.status_code}"
3379     except requests.exceptions.RequestException as e:
3380         return False, str(e)
3381
3382 def check_system_status():
3383     print(f"\n{Colors.BOLD}Local RAG System Status{Colors.ENDC}\n")
3384
3385     # Docker services
3386     print(f"{Colors.BOLD}Docker Services:{Colors.ENDC}")
3387     docker_services = [

```



```

3385         ("vector-db", "Qdrant Vector DB"),
3386         ("mlflow", "MLflow Server"),
3387         ("flask-app", "Flask Web App")
3388     ]
3389
3390     for service_id, service_name in docker_services:
3391         status, message = check_docker_service(service_id)
3392         print_status(service_name, status, message)
3393
3394     # HTTP endpoints
3395     print(f"\n{Colors.BOLD}Service Endpoints:{Colors.ENDC}")
3396     endpoints = [
3397         ("http://localhost:6333/health", "Vector DB API"),
3398         ("http://localhost:5001/ping", "MLflow API"),
3399         ("http://localhost:8000/api/health", "Flask API"),
3400     ]
3401
3402     for url, name in endpoints:
3403         status, message = check_http_endpoint(url)
3404         print_status(name, status, message)
3405
3406     # MLflow model
3407     print(f"\n{Colors.BOLD}MLflow Model:{Colors.ENDC}")
3408     try:
3409         # Check if the flask API can communicate with MLflow
3410         response = requests.get("http://localhost:8000/api/health", timeout=2)
3411         if response.status_code == 200:
3412             data = response.json()
3413             mlflow_status = data.get("mlflow", False)
3414             print_status("MLflow Model", mlflow_status,
3415                         "Model is deployed and ready" if mlflow_status else "Model not
3416 deployed or not responding")
3417         else:
3418             print_status("MLflow Model", False, "Could not check via Flask API")
3419     except:
3420         print_status("MLflow Model", False, "Could not check via Flask API")
3421
3422     # Data and Model directories
3423     print(f"\n{Colors.BOLD}Data Directories:{Colors.ENDC}")
3424     directories = [
3425         ("./data/pdfs", "PDF Storage"),
3426         ("./data/vectors", "Vector Storage"),
3427         ("./models/embedding", "Embedding Model"),
3428         ("./models/reranker", "Reranker Model"),
3429         ("./models/llm", "LLM Model")
3430     ]
3431
3432     for path, name in directories:
3433         exists = os.path.exists(path)
3434         if exists and os.path.isdir(path):

```

```

3432         items = len(os.listdir(path))
3433         print_status(name, True, f"{items} items")
3434     else:
3435         print_status(name, False, "Directory not found")
3436
3437 def start_service(service_name):
3438     print(f"Starting {service_name}...")
3439     try:
3440         if service_name == "all":
3441             subprocess.run(["docker-compose", "up", "-d"], check=True)
3442             print("All Docker services started")
3443
3444             # Start the MLflow model server
3445             print("Starting MLflow model server...")
3446             subprocess.Popen(["python", "app/scripts/deploy_model.py"],
3447                             stdout=subprocess.PIPE, stderr=subprocess.PIPE)
3448
3449         elif service_name in ["vector-db", "mlflow", "flask-app"]:
3450             subprocess.run(["docker-compose", "up", "-d", service_name], check=True)
3451             print(f"{service_name} started")
3452
3453         elif service_name == "mlflow-model":
3454             subprocess.run(["python", "app/scripts/deploy_model.py"], check=True)
3455             print("MLflow model deployed")
3456
3457         else:
3458             print(f"Unknown service: {service_name}")
3459             return
3460
3461         print("Waiting for service to be ready...")
3462         time.sleep(5)
3463         check_system_status()
3464
3465     except subprocess.CalledProcessError as e:
3466         print(f"Error starting service: {e}")
3467
3468 def stop_service(service_name):
3469     print(f"Stopping {service_name}...")
3470     try:
3471         if service_name == "all":
3472             subprocess.run(["docker-compose", "down"], check=True)
3473             print("All Docker services stopped")
3474
3475         elif service_name in ["vector-db", "mlflow", "flask-app"]:
3476             subprocess.run(["docker-compose", "stop", service_name], check=True)
3477             print(f"{service_name} stopped")
3478
3479         elif service_name == "mlflow-model":
3480             print("MLflow model server cannot be stopped directly")
3481             print("To stop it, restart the MLflow Docker container:")

```

```

3479         print(" docker-compose restart mlflow")
3480
3481     else:
3482         print(f"Unknown service: {service_name}")
3483         return
3484
3485     check_system_status()
3486
3487 except subprocess.CalledProcessError as e:
3488     print(f"Error stopping service: {e}")
3489
3490 if __name__ == "__main__":
3491     parser = argparse.ArgumentParser(description="Check status of Local RAG System")
3492     parser.add_argument("--start", help="Start a service (vector-db, mlflow, flask-app, mlflow-model, all)")
3493     parser.add_argument("--stop", help="Stop a service (vector-db, mlflow, flask-app, all)")
3494
3495     args = parser.parse_args()
3496
3497     if args.start:
3498         start_service(args.start)
3499     elif args.stop:
3500         stop_service(args.stop)
3501     else:
3502         check_system_status()

```

**Professor's Hint:** *Integration testing is critical for complex systems with multiple components. The system status check script helps users quickly diagnose problems and restart specific components if needed. Make this script easily accessible and user-friendly to encourage its use.*

*Consistency Testing.* To ensure the system components work together consistently, let's add a few integration tests that specifically check for compatibility between components:

1. Create a component integration test:

```
touch app/tests/test_component_integration.py
```

2. Add the following to app/tests/test\_component\_integration.py:

```

3514 import os
3515 import sys
3516 import pytest
3517 import numpy as np
3518 from pathlib import Path
3519
3520 # Add the project root to the Python path
3521 sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
3522
3523 from app.config.settings import (
3524     EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH, LLM_MODEL_PATH,
3525     VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION

```

```

3526 )
3527 from app.utils.query_processing import QueryProcessor
3528 from app.utils.reranking import Reranker
3529 from app.utils.vector_db import VectorDBClient
3530 from app.utils.llm import LLMProcessor
3531
3532 class TestComponentIntegration:
3533     """Test the integration between different components."""
3534
3535     def test_embedding_dimensions(self):
3536         """Test that embedding dimensions are consistent across components."""
3537         # Load query processor (which uses the embedding model)
3538         query_processor = QueryProcessor(EMBEDDING_MODEL_PATH)
3539
3540         # Generate a test embedding
3541         test_query = "This is a test query"
3542         query_embedding = query_processor.process_query(test_query)
3543
3544         # Check dimension
3545         assert query_embedding.shape[0] == VECTOR_DIMENSION, \
3546             f"Embedding dimension ({query_embedding.shape[0]}) doesn't match configured dimension ({VECTOR_DIMENSION})"
3547
3548         # Check if vector DB is configured with same dimension
3549         vector_db = VectorDBClient(VECTOR_DB_HOST, VECTOR_DB_PORT, "test_consistency",
3550             VECTOR_DIMENSION)
3551
3552         # Try creating a collection and check if it accepts the embedding
3553         vector_db.create_collection()
3554
3555         # Clean up
3556         vector_db.delete_collection()
3557
3558     def test_reranker_compatibility(self):
3559         """Test that reranker can process outputs from vector search."""
3560         # Create test data
3561         test_query = "This is a test query"
3562         test_results = [
3563             {
3564                 'chunk_id': 'test_chunk_1',
3565                 'chunk_text': 'This is the first test chunk',
3566                 'score': 0.95
3567             },
3568             {
3569                 'chunk_id': 'test_chunk_2',
3570                 'chunk_text': 'This is the second test chunk',
3571                 'score': 0.85
3572             }
3573         ]

```

```

3573     # Load reranker
3574     reranker = Reranker(RERANKER_MODEL_PATH)
3575
3576     # Try reranking results
3577     reranked_results = reranker.rerank(test_query, test_results)
3578
3579     # Check if reranking worked
3580     assert len(reranked_results) == len(test_results), "Reranker changed the number of
results"
3581     assert 'rerank_score' in reranked_results[0], "Reranker did not add scores"
3582
3583 def test_llm_prompt_compatibility(self):
3584     """Test that LLM can process prompts created from reranked results."""
3585     # Skip if LLM model doesn't exist
3586     if not os.path.exists(LLM_MODEL_PATH):
3587         pytest.skip(f"LLM model not found at {LLM_MODEL_PATH}")
3588
3589     # Create test data
3590     test_query = "This is a test query"
3591     test_results = [
3592         {
3593             'chunk_id': 'test_chunk_1',
3594             'chunk_text': 'This is the first test chunk with important information.',
3595             'rerank_score': 0.95,
3596             'score': 0.90
3597         },
3598         {
3599             'chunk_id': 'test_chunk_2',
3600             'chunk_text': 'This is the second test chunk with different information.',
3601             'rerank_score': 0.85,
3602             'score': 0.80
3603         }
3604     ]
3605
3606     # Load LLM processor
3607     llm_processor = LLMProcessor(LLM_MODEL_PATH, context_size=1024, max_tokens=100)
3608
3609     # Create prompt
3610     prompt = llm_processor.create_prompt(test_query, test_results)
3611
3612     # Check prompt structure
3613     assert test_query in prompt, "Query not found in prompt"
3614     assert test_results[0]['chunk_text'] in prompt, "Context not found in prompt"
3615
3616     # Optional: Test actual generation if environment allows
3617     try:
3618         response = llm_processor.generate_response(prompt)
3619         assert 'text' in response, "Response missing text field"
3620         assert 'metadata' in response, "Response missing metadata field"
3621     except Exception as e:

```

```

3620         pytest.skip(f"LLM generation test skipped: {str(e)}")
3621
3622 if __name__ == "__main__":
3623     # Run tests
3624     pytest.main(["-xvs", __file__])

```

This test specifically focuses on ensuring that the dimensions, data formats, and interfaces between components are consistent, which is critical for system integration.

*Lab 6.3: System Consistency Verification.*

*Lab 6.2: Offline Mode Testing.*

*Learning Objectives:*

- Test the system with internet connectivity disabled
- Identify and fix dependencies on external services
- Ensure data persistence across system restarts

*Exercises:*

1. Create an offline mode test script:

```

3639 touch offline_test.sh
3640 chmod +x offline_test.sh
3641

```

2. Add the following to offline\_test.sh:

```

3644 #!/bin/bash
3645
3646 # Colors for output
3647 GREEN='\033[0;32m'
3648 YELLOW='\033[1;33m'
3649 RED='\033[0;31m'
3650 NC='\033[0m' # No Color
3651
3652 echo -e "${YELLOW}Offline Mode Test${NC}"
3653 echo "This script will test if the system works properly without internet connectivity."
3654 echo "It will temporarily disable network access for Docker containers."
3655
3656 # Check if script is run as root
3657 if [ "$EUID" -ne 0 ]; then
3658     echo -e "${RED}Please run as root to modify network settings${NC}"
3659     exit 1
3660 fi
3661
3662 # Check if system is running
3663 echo "Checking if system is running..."
3664 if ! docker ps | grep -q "vector-db"; then
3665     echo -e "${RED}Vector database container not running. Please start the system first.${NC}"
3666     exit 1
3667 fi

```

```

3667
3668 if ! docker ps | grep -q "mlflow"; then
3669     echo -e "${RED}MLflow container not running. Please start the system first.${NC}"
3670     exit 1
3671 fi
3672
3673 if ! docker ps | grep -q "flask-app"; then
3674     echo -e "${RED}Flask app container not running. Please start the system first.${NC}"
3675     exit 1
3676 fi
3677
3678 echo -e "${GREEN}All containers are running.${NC}"
3679
3680 # Create a Docker network with no internet access
3681 echo "Creating isolated Docker network..."
3682 docker network create --internal isolated-network
3683
3684 # Move containers to isolated network
3685 echo "Moving containers to isolated network..."
3686 docker network connect isolated-network $(docker ps -q --filter name=vector-db)
3687 docker network connect isolated-network $(docker ps -q --filter name=mlflow)
3688 docker network connect isolated-network $(docker ps -q --filter name=flask-app)
3689
3690 # Disconnect from default network
3691 echo "Disconnecting from default network..."
3692 docker network disconnect bridge $(docker ps -q --filter name=vector-db)
3693 docker network disconnect bridge $(docker ps -q --filter name=mlflow)
3694 docker network disconnect bridge $(docker ps -q --filter name=flask-app)
3695
3696 echo -e "${YELLOW}Containers are now in offline mode.${NC}"
3697 echo "Testing system functionality..."
3698
3699 # Test if services are still responding
3700 echo "Testing vector database..."
3701 if curl -s http://localhost:6333/health > /dev/null; then
3702     echo -e "${GREEN}Vector database is responding in offline mode.${NC}"
3703 else
3704     echo -e "${RED}Vector database is not responding!${NC}"
3705 fi
3706
3707 echo "Testing MLflow..."
3708 if curl -s http://localhost:5001/ping > /dev/null; then
3709     echo -e "${GREEN}MLflow is responding in offline mode.${NC}"
3710 else
3711     echo -e "${RED}MLflow is not responding!${NC}"
3712 fi
3713
3714 echo "Testing Flask app..."
3715 if curl -s http://localhost:8000/api/health > /dev/null; then
3716     echo -e "${GREEN}Flask app is responding in offline mode.${NC}"
3717

```

```

3714 else
3715     echo -e "${RED}Flask app is not responding!${NC}"
3716 fi
3717
3718 # Test a query to verify end-to-end functionality
3719 echo "Testing query functionality..."
3720 QUERY_RESULT=$(curl -s -X POST -H "Content-Type: application/json" -d '{"question":"What is
3721     machine learning?"}' http://localhost:8000/api/ask)
3722
3723 if [[ $QUERY_RESULT == *"text"* ]]; then
3724     echo -e "${GREEN}Query functionality is working in offline mode.${NC}"
3725 else
3726     echo -e "${RED}Query functionality is not working in offline mode!${NC}"
3727     echo "Response: $QUERY_RESULT"
3728 fi
3729
3730 # Wait for user to check the system
3731 echo -e "${YELLOW}The system is now in offline mode. You can test it manually.${NC}"
3732 echo "Press Enter to restore network connectivity..."
3733 read
3734
3735 # Restore network connectivity
3736 echo "Restoring network connectivity..."
3737 docker network connect bridge $(docker ps -q --filter name=vector-db)
3738 docker network connect bridge $(docker ps -q --filter name=mlflow)
3739 docker network connect bridge $(docker ps -q --filter name=flask-app)
3740
3741 docker network disconnect isolated-network $(docker ps -q --filter name=vector-db)
3742 docker network disconnect isolated-network $(docker ps -q --filter name=mlflow)
3743 docker network disconnect isolated-network $(docker ps -q --filter name=flask-app)
3744
3745 # Remove isolated network
3746 docker network rm isolated-network
3747
3748 echo -e "${GREEN}Network connectivity restored.${NC}"
3749 echo "Offline mode test completed."

```

### 3. Create a backup and restore script:

```

touch backup_restore.sh
chmod +x backup_restore.sh

```

### 4. Add the following to backup\_restore.sh:

```

#!/bin/bash

# Colors for output
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
RED='\033[0;31m'

```



```

3761 NC='\033[0m' # No Color
3762
3763 # Default backup directory
3764 BACKUP_DIR="./backups"
3765 mkdir -p "$BACKUP_DIR"
3766
3767 function backup() {
3768     local timestamp=$(date +%Y%m%d_%H%M%S)
3769     local backup_file="${BACKUP_DIR}/rag_backup_${timestamp}.tar.gz"
3770
3771     echo -e "${YELLOW}Creating backup...${NC}"
3772
3773     # Stop services to ensure data consistency
3774     echo "Stopping services..."
3775     docker-compose stop
3776
3777     # Create backup
3778     echo "Creating backup archive..."
3779     tar -czf "$backup_file" \
3780         --exclude="venv" \
3781         --exclude="__pycache__" \
3782         --exclude=".git" \
3783         --exclude="*.log" \
3784         --exclude="*.tar.gz" \
3785         --exclude="mlflow/artifacts/*/tmp" \
3786         ./data ./models ./mlflow ./app ./flask-app
3787
3788     # Restart services
3789     echo "Restarting services..."
3790     docker-compose up -d
3791
3792     echo -e "${GREEN}Backup created: ${backup_file}${NC}"
3793     echo "You can restore this backup using:"
3794     echo ". /backup_restore.sh restore ${backup_file}"
3795 }
3796
3797 function restore() {
3798     local backup_file=$1
3799
3800     if [ ! -f "$backup_file" ]; then
3801         echo -e "${RED}Backup file not found: ${backup_file}${NC}"
3802         exit 1
3803     fi
3804
3805     echo -e "${YELLOW}Restoring from backup: ${backup_file}${NC}"
3806
3807     # Stop services
3808     echo "Stopping services..."
3809     docker-compose stop

```

```

3808 # Create temporary directory
3809 local temp_dir=$(mktemp -d)
3810
3811 # Extract backup
3812 echo "Extracting backup..."
3813 tar -xzf "$backup_file" -C "$temp_dir"
3814
3815 # Restore data
3816 echo "Restoring data..."
3817 rsync -a --delete "$temp_dir/data/" ./data/
3818 rsync -a --delete "$temp_dir/models/" ./models/
3819 rsync -a --delete "$temp_dir/mlflow/" ./mlflow/
3820
3821 # Clean up
3822 rm -rf "$temp_dir"
3823
3824 # Restart services
3825 echo "Restarting services..."
3826 docker-compose up -d
3827
3828 echo -e "${GREEN}Restore completed.${NC}"
3829 }
3830
3831 function list_backups() {
3832     echo -e "${YELLOW}Available backups:${NC}"
3833
3834     local count=0
3835     for file in "$BACKUP_DIR"/rag_backup_*.tar.gz; do
3836         if [ -f "$file" ]; then
3837             local size=$(du -h "$file" | cut -f1)
3838             local date=$(stat -c %y "$file" | cut -d. -f1)
3839             echo "$(basename "$file") (${size}, ${date})"
3840             count=$((count + 1))
3841         fi
3842     done
3843
3844     if [ $count -eq 0 ]; then
3845         echo "No backups found."
3846     fi
3847 }
3848
3849 # Main script
3850 case "$1" in
3851     backup)
3852         backup
3853         ;;
3854     restore)
3855         if [ -z "$2" ]; then
3856             echo -e "${RED}Error: No backup file specified.${NC}"
3857             echo "Usage: $0 restore <backup_file>"
3858         fi
3859     ;;
3860 *)
3861     echo -e "${RED}Invalid command.${NC}"
3862     ;;
3863 esac

```

```

3855         exit 1
3856     fi
3857     restore "$2"
3858 ;;
3859 list)
3860     list_backups
3861 ;;
3862 *)
3863     echo "Usage: $0 {backup|restore|list}"
3864     echo "  backup      Create a new backup"
3865     echo "  restore <file> Restore from backup file"
3866     echo "  list        List available backups"
3867     exit 1
3868 ;;
3869 esac

```

**Professor's Hint:** Regular backups are essential for any system that stores important data. The backup script not only creates archives of your data and models but also ensures data consistency by stopping services before backing up and restarting them afterward.

#### Checkpoint Questions:

1. What happens if one of the Docker containers fails during system operation?
2. How can we make the system resilient to temporary failures?
3. Why is testing in offline mode important for a locally deployed RAG system?
4. What failure modes are unique to RAG systems compared to traditional search or pure LLM applications?
5. How would you implement graceful degradation if one component of the system fails?
6. What would be the most critical components to monitor in a production RAG system?
7. How could you design the system architecture to allow for horizontal scaling if processing demands increase?
8. What security considerations should be addressed for a system that processes potentially sensitive documents?
9. **Additional Challenge:** Implement a cron job to automatically create daily backups and retain only the 7 most recent backups.

#### Final Project Submission Guidelines

##### Requirements.

1. **Complete System**
  - All components integrated and working together
  - Fully functional in offline mode
  - Clear documentation for usage
2. **Technical Report (3-5 pages)**
  - System architecture overview
  - Component descriptions
  - Performance analysis
  - Limitations and future improvements

### 3. Code Repository

- Well-organized codebase
- Proper comments and docstrings
- README with setup and usage instructions
- Requirements file with all dependencies

### 4. Demonstration Video (5-7 minutes)

- System walkthrough
- Example usage scenarios
- Performance demonstration

*Self-Assessment Questions.* Before submitting your project, ask yourself:

1. Can the system function completely offline after initial setup?
2. Does the RAG system correctly extract text from various PDF formats?
3. Is the vector search accurate and relevant to user queries?
4. Does the LLM generate coherent and factual responses based on the retrieved context?
5. Is the web interface intuitive and responsive?
6. Does the system handle errors gracefully?
7. Is there proper documentation for all components?
8. Can another person set up and use your system based on your documentation?

## Conclusion

Building a local RAG system requires integration of multiple complex components, from PDF processing and vector embeddings to large language models and web interfaces. This project has given you hands-on experience with the entire pipeline, providing a foundation for developing more advanced AI applications.

The skills you've developed can be applied to many other domains, including: - Custom knowledge bases for specific domains - Intelligent document search systems - Automated document analysis - Personalized AI assistants

Remember that local AI systems have unique advantages in terms of privacy, data control, and offline functionality. As the field continues to evolve, the ability to deploy AI systems locally will become increasingly valuable for many applications.

Good luck with your final project!

**Professor's Final Hint:** *The true test of any system is how it performs with real-world data and users. Take time to test your system with diverse PDFs and questions, and iterate based on what you learn. The most valuable systems are those that solve real problems elegantly and reliably.*

## Lab 4: MLflow Integration and Model Serving

### Lab 4.1: MLflow Model Wrapper and Logging.

*Learning Objectives:*

- Create an MLflow model wrapper for the RAG system
- Log models and artifacts to MLflow
- Understand model versioning and management

*Exercises:*

1. Create an MLflow model wrapper:

```
mkdir -p app/models
touch app/models/rag_model.py
```

2. Add the following to app/models/rag\_model.py:

```
import os
import sys
import pandas as pd
import numpy as np
from typing import Dict, Any
import mlflow.pyfunc
import logging

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

class RAGModel(mlflow.pyfunc.PythonModel):
    def __init__(self):
        """Initialize the RAG model wrapper."""
        self.rag_processor = None

    def load_context(self, context):
        """
        Load model artifacts.

        Args:
            context: MLflow model context
        """
        logger.info("Loading RAG model context")

        # Add paths
        sys.path.append(os.path.dirname(context.artifacts['app_dir']))

        # Import here to avoid circular imports
        from app.config.settings import (
            VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
            EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH, LLM_MODEL_PATH
        )

        from app.utils.search import create_search_pipeline
        from app.utils.llm import create_rag_processor

        # Create search pipeline
        search_pipeline = create_search_pipeline(
            VECTOR_DB_HOST, VECTOR_DB_PORT, COLLECTION_NAME, VECTOR_DIMENSION,
            EMBEDDING_MODEL_PATH, RERANKER_MODEL_PATH
```

```

3996         )
3997
3998         # Create RAG processor
3999         self.rag_processor = create_rag_processor(search_pipeline, LLM_MODEL_PATH)
4000
4001         logger.info("RAG model context loaded")
4002
4003     def predict(self, context, model_input):
4004         """
4005         Generate predictions.
4006
4007         Args:
4008             context: MLflow model context
4009             model_input: Input data
4010
4011         Returns:
4012             Model predictions
4013         """
4014         # Check if input is a pandas DataFrame
4015         if isinstance(model_input, pd.DataFrame):
4016             # Extract query
4017             if 'query' in model_input.columns:
4018                 query = model_input['query'].iloc[0]
4019             else:
4020                 raise ValueError("Input DataFrame must have a 'query' column")
4021         elif isinstance(model_input, dict):
4022             # Extract query from dictionary
4023             if 'query' in model_input:
4024                 query = model_input['query']
4025             else:
4026                 raise ValueError("Input dictionary must have a 'query' key")
4027         else:
4028             # Assume input is a string query
4029             query = str(model_input)
4030
4031         logger.info(f"Processing query: {query}")
4032
4033         # Process query
4034         response = self.rag_processor.process_query(query)
4035
4036         return response
4037
4038     def get_pip_requirements():
4039         """Get pip requirements for the model."""
4040         return [
4041             "pandas",
4042             "numpy",
4043             "scikit-learn",
4044             "sentence-transformers",
4045             "qdrant-client",

```

```

4043         "llama-cpp-python",
4044         "mlflow"
4045     ]
4046
4047 if __name__ == "__main__":
4048     # Test the model
4049     model = RAGModel()
4050
4051     # Create a dummy context
4052     class DummyContext:
4053         def __init__(self):
4054             self.artifacts = {'app_dir':
4055                 os.path.dirname(os.path.dirname(os.path.abspath(__file__)))}
4056
4057     # Load context
4058     model.load_context(DummyContext())
4059
4060     # Test prediction
4061     result = model.predict(None, "What is retrieval-augmented generation?")
4062     print(result['text'])

```

3. Create a script to log the model to MLflow:

```
touch app/scripts/log_model.py
```

4. Add the following to app/scripts/log\_model.py:

```

4068 import os
4069 import sys
4070 import mlflow
4071 import logging
4072 from pathlib import Path
4073
4074 # Configure logging
4075 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
4076     %(message)s')
4077 logger = logging.getLogger(__name__)
4078
4079 # Add the project root to the Python path
4080 sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
4081
4082 from app.config.settings import MLFLOW_TRACKING_URI, MLFLOW_MODEL_NAME
4083 from app.models.rag_model import RAGModel, get_pip_requirements
4084
4085 def log_model():
4086     """Log the RAG model to MLflow."""
4087     logger.info(f"Logging model {MLFLOW_MODEL_NAME} to MLflow at {MLFLOW_TRACKING_URI}")
4088
4089     # Set MLflow tracking URI
4090     mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)

```

```

4090
4091 # Get project root
4092 project_root = Path(__file__).resolve().parent.parent.parent
4093 app_dir = os.path.join(project_root, "app")
4094
4095 # Log model
4096 with mlflow.start_run(run_name=f"{MLFLOW_MODEL_NAME}_deployment") as run:
4097     # Log parameters
4098     mlflow.log_param("embedding_model", "all-MiniLM-L6-v2")
4099     mlflow.log_param("reranker_model", "ms-marco-MiniLM-L-6-v2")
4100     mlflow.log_param("llm_model", "llama-2-7b-chat-q4_0.gguf")
4101
4102     # Log model
4103     model_info = mlflow.pyfunc.log_model(
4104         artifact_path="model",
4105         python_model=RAGModel(),
4106         artifacts={
4107             "app_dir": app_dir
4108         },
4109         pip_requirements=get_pip_requirements(),
4110         registered_model_name=MLFLOW_MODEL_NAME
4111     )
4112
4113     logger.info(f"Model logged: {model_info.model_uri}")
4114
4115     return model_info
4116
4117 if __name__ == "__main__":
4118     # Log model
4119     model_info = log_model()
4120     print(f"Model logged: {model_info.model_uri}")
4121     print(f"Run ID: {model_info.run_id}")

```

**Professor's Hint:** MLflow's Python Function (PyFunc) model format is very flexible but doesn't handle complex dependencies well. By including the `app_dir` as an artifact, we ensure that all necessary code is available, but this approach requires the filesystem structure to be preserved. For production, consider packaging your RAG components as a Python package.

5. Create a script to deploy the model:

```
touch app/scripts/deploy_model.py
```

6. Add the following to `app/scripts/deploy_model.py`:

```

4130 import os
4131 import sys
4132 import mlflow
4133 import argparse
4134 import logging
4135 from pathlib import Path
4136

```



```

4137
4138 # Configure logging
4139 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
4140                      %(message)s')
4141 logger = logging.getLogger(__name__)
4142
4143 # Add the project root to the Python path
4144 sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
4145
4146 from app.config.settings import MLFLOW_TRACKING_URI, MLFLOW_MODEL_NAME
4147
4148 def deploy_model(run_id=None, port=5001):
4149     """
4150     Deploy the RAG model using MLflow.
4151
4152     Args:
4153         run_id: Run ID to deploy (if None, use latest version)
4154         port: Port to deploy on
4155     """
4156     # Set MLflow tracking URI
4157     mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)
4158
4159     # Get model URI
4160     if run_id:
4161         model_uri = f"runs:{run_id}/model"
4162         logger.info(f"Deploying model from run {run_id}")
4163     else:
4164         model_uri = f"models:{MLFLOW_MODEL_NAME}/latest"
4165         logger.info(f"Deploying latest version of model {MLFLOW_MODEL_NAME}")
4166
4167     # Deploy model
4168     logger.info(f"Starting MLflow serving on port {port}")
4169     os.system(f"mlflow models serve -m {model_uri} -p {port} --no-conda")
4170
4171 if __name__ == "__main__":
4172     # Parse arguments
4173     parser = argparse.ArgumentParser(description='Deploy the RAG model')
4174     parser.add_argument('--run-id', type=str, default=None,
4175                         help='Run ID to deploy')
4176     parser.add_argument('--port', type=int, default=5001,
4177                         help='Port to deploy on')
4178     args = parser.parse_args()
4179
4180     # Deploy model
4181     deploy_model(args.run_id, args.port)

```

### Checkpoint Questions:

1. Why do we use MLflow for model versioning and deployment?

2. What are the advantages of packaging a model with MLflow compared to directly running the application?
3. How does the MLflow model server handle inference requests?
4. How would you design an A/B testing framework to evaluate changes to different components of the RAG pipeline?
5. What monitoring metrics would be most important for understanding RAG system performance in production?
6. How does the choice of serialization format for model artifacts affect the tradeoff between load time and storage efficiency?
7. What strategies could you implement to handle model updates without service interruption?
8. **Additional Challenge:** Create a system for A/B testing different model configurations by deploying multiple versions of the model and comparing their performance.

#### Lab 4.2: MLflow Client and Integration Testing.

##### Learning Objectives:

- Create a client to interact with the MLflow serving endpoint
- Implement integration testing for the end-to-end system
- Test system performance and reliability

##### Exercises:

1. Create an MLflow client utility:

```
mkdir -p app/clients
touch app/clients/mlflow_client.py
```

2. Add the following to app/clients/mlflow\_client.py:

```
import os
import json
import requests
from typing import Dict, Any, Union
import logging

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

class MLflowClient:
    def __init__(self, endpoint_url: str):
        """
        Initialize the MLflow client.

        Args:
            endpoint_url: URL of the MLflow serving endpoint
        """
        self.endpoint_url = endpoint_url
        logger.info(f"Initialized MLflow client for endpoint: {endpoint_url}")
```

```

4231
4232 def predict(self, query: str) -> Dict[str, Any]:
4233     """
4234     Make a prediction using the MLflow serving endpoint.
4235
4236     Args:
4237         query: Query text
4238
4239     Returns:
4240         Prediction result
4241     """
4242     logger.info(f"Sending query to MLflow endpoint: {query}")
4243
4244     # Create payload
4245     payload = {
4246         "query": query
4247     }
4248
4249     # Send request
4250     response = requests.post(
4251         f"{self.endpoint_url}/invocations",
4252         json=payload,
4253         headers={"Content-Type": "application/json"}
4254     )
4255
4256     # Check response
4257     if response.status_code != 200:
4258         logger.error(f"Error from MLflow endpoint: {response.text}")
4259         raise Exception(f"Error from MLflow endpoint: {response.text}")
4260
4261     # Parse response
4262     result = response.json()
4263
4264     logger.info(f"Received response from MLflow endpoint")
4265     return result
4266
4267 def is_alive(self) -> bool:
4268     """
4269     Check if the MLflow endpoint is alive.
4270
4271     Returns:
4272         True if the endpoint is alive, False otherwise
4273     """
4274     try:
4275         response = requests.get(f"{self.endpoint_url}/ping")
4276         return response.status_code == 200
4277     except:
4278         return False
4279
4280 def create_mlflow_client(host: str = "localhost", port: int = 5001) -> MLflowClient:

```

```

4278 """
4279 Create an MLflow client.
4280
4281 Args:
4282     host: Host of the MLflow server
4283     port: Port of the MLflow server
4284
4285 Returns:
4286     MLflow client
4287 """
4288 endpoint_url = f"http://{host}:{port}"
4289 return MLflowClient(endpoint_url)
4290
4291 if __name__ == "__main__":
4292     # Example usage
4293     client = create_mlflow_client()
4294
4295     # Check if endpoint is alive
4296     if client.is_alive():
4297         print("MLflow endpoint is alive")
4298
4299     # Make a prediction
4300     response = client.predict("What is retrieval-augmented generation?")
4301     print(f"Response: {response['text']}")
4302
4303 else:
4304     print("MLflow endpoint is not available. Make sure the model is deployed.")

```

### 3. Create an integration test script:

```
touch app/tests/test_integration.py
```

### 4. Add the following to app/tests/test\_integration.py:

```

4308 import os
4309 import sys
4310 import time
4311 import pytest
4312 from pathlib import Path
4313
4314 # Add the project root to the Python path
4315 sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
4316
4317 from app.clients.mlflow_client import create_mlflow_client
4318
4319 @pytest.fixture
4320 def mlflow_client():
4321     """Create an MLflow client for testing."""
4322     client = create_mlflow_client()
4323     if not client.is_alive():
4324         pytest.skip("MLflow endpoint is not available. Make sure the model is deployed.")

```

```

4325     return client
4326
4327 def test_mlflow_endpoint_alive(mlflow_client):
4328     """Test that the MLflow endpoint is alive."""
4329     assert mlflow_client.is_alive(), "MLflow endpoint is not alive"
4330
4331 def test_simple_query(mlflow_client):
4332     """Test a simple query."""
4333     response = mlflow_client.predict("What is machine learning?")
4334     assert 'text' in response, "Response missing 'text' field"
4335     assert len(response['text']) > 0, "Response text is empty"
4336     assert 'sources' in response, "Response missing 'sources' field"
4337     assert 'metadata' in response, "Response missing 'metadata' field"
4338
4339 def test_query_with_no_results(mlflow_client):
4340     """Test a query that should not have results in the corpus."""
4341     response = mlflow_client.predict("What is the capital of Jupiter?")
4342     assert 'text' in response, "Response missing 'text' field"
4343     # The response should indicate that the information is not available
4344     assert len(response['text']) > 0, "Response text is empty"
4345
4346 def test_response_timing(mlflow_client):
4347     """Test the response time of the endpoint."""
4348     query = "What is retrieval-augmented generation?"
4349
4350     # Measure response time
4351     start_time = time.time()
4352     response = mlflow_client.predict(query)
4353     end_time = time.time()
4354
4355     response_time = end_time - start_time
4356     print(f"Response time: {response_time:.2f} seconds")
4357
4358     # We expect a response in under 10 seconds for a simple query
4359     assert response_time < 10, f"Response time too slow: {response_time:.2f} seconds"
4360
4361 def test_multiple_queries(mlflow_client):
4362     """Test multiple consecutive queries."""
4363     queries = [
4364         "What is vector search?",
4365         "How does re-ranking work?",
4366         "What are embeddings?",
4367         "How does Llama 2 compare to other language models?"
4368     ]
4369
4370     for query in queries:
4371         response = mlflow_client.predict(query)
4372         assert 'text' in response, f"Response missing 'text' field for query: {query}"
4373         assert len(response['text']) > 0, f"Response text is empty for query: {query}"

```

```

4372 if __name__ == "__main__":
4373     # Run tests
4374     pytest.main(["-xvs", __file__])

```

**Professor’s Hint:** *Integration tests are essential for ensuring the reliability of complex systems like this RAG application. The tests should cover both “happy path” scenarios and edge cases. Pay special attention to response timing if the system is too slow, users will find it frustrating regardless of accuracy.*

5. Create a load testing script (optional):

```

4381 touch app/tests/load_test.py

```

6. Add the following to `app/tests/load_test.py`:

```

4385 import os
4386 import sys
4387 import time
4388 import random
4389 import threading
4390 import concurrent.futures
4391 from pathlib import Path

4392 # Add the project root to the Python path
4393 sys.path.insert(0, str(Path(__file__).resolve().parent.parent.parent))
4394
4395 from app.clients.mlflow_client import create_mlflow_client

4396 # Test queries
4397 TEST_QUERIES = [
4398     "What is retrieval-augmented generation?",
4399     "How do vector databases work?",
4400     "What is semantic search?",
4401     "How do transformers work?",
4402     "What is the difference between bi-encoders and cross-encoders?",
4403     "How does Llama 2 compare to other language models?",
4404     "What is prompt engineering?",
4405     "How do you evaluate RAG systems?",
4406     "What is the role of re-ranking in search?",
4407     "How do embeddings capture semantic meaning?",
4408 ]

4409 def run_query(client, query):
4410     """Run a query and return the response time."""
4411     start_time = time.time()
4412     try:
4413         response = client.predict(query)
4414         success = True
4415     except Exception as e:
4416         print(f"Error: {str(e)}")
4417         success = False

```

```

4419     end_time = time.time()
4420
4421     return {
4422         'query': query,
4423         'response_time': end_time - start_time,
4424         'success': success,
4425         'timestamp': time.time()
4426     }
4427
4428 def worker(client, num_queries, results):
4429     """Worker function for concurrent queries."""
4430     for _ in range(num_queries):
4431         query = random.choice(TEST_QUERIES)
4432         result = run_query(client, query)
4433         results.append(result)
4434         time.sleep(random.uniform(0.5, 2.0)) # Random delay between queries
4435
4436 def run_load_test(num_workers=3, queries_per_worker=5):
4437     """
4438     Run a load test with multiple concurrent workers.
4439
4440     Args:
4441         num_workers: Number of concurrent workers
4442         queries_per_worker: Number of queries per worker
4443     """
4444     print(f"Running load test with {num_workers} workers, {queries_per_worker} queries per worker")
4445
4446     # Create client
4447     client = create_mlflow_client()
4448
4449     # Check if endpoint is alive
4450     if not client.is_alive():
4451         print("MLflow endpoint is not available. Make sure the model is deployed.")
4452         return
4453
4454     # Shared results list
4455     results = []
4456
4457     # Run workers in parallel
4458     start_time = time.time()
4459     with concurrent.futures.ThreadPoolExecutor(max_workers=num_workers) as executor:
4460         futures = [
4461             executor.submit(worker, client, queries_per_worker, results)
4462             for _ in range(num_workers)
4463         ]
4464         concurrent.futures.wait(futures)
4465     end_time = time.time()
4466
4467     # Analyze results

```

```

4466 total_queries = len(results)
4467 successful_queries = sum(1 for r in results if r['success'])
4468 failed_queries = total_queries - successful_queries
4469
4470 if total_queries > 0:
4471     avg_response_time = sum(r['response_time'] for r in results) / total_queries
4472     max_response_time = max(r['response_time'] for r in results)
4473     min_response_time = min(r['response_time'] for r in results)
4474 else:
4475     avg_response_time = max_response_time = min_response_time = 0
4476
4477 total_time = end_time - start_time
4478 queries_per_second = total_queries / total_time if total_time > 0 else 0
4479
4480 # Print results
4481 print("\nLoad Test Results")
4482 print("-----")
4483 print(f"Total queries: {total_queries}")
4484 print(f"Successful queries: {successful_queries}")
4485 print(f"Failed queries: {failed_queries}")
4486 print(f"Success rate: {successful_queries / total_queries * 100:.2f}%")
4487 print(f"Average response time: {avg_response_time:.2f} seconds")
4488 print(f"Min response time: {min_response_time:.2f} seconds")
4489 print(f"Max response time: {max_response_time:.2f} seconds")
4490 print(f"Total test time: {total_time:.2f} seconds")
4491 print(f"Queries per second: {queries_per_second:.2f}")
4492
4493 if __name__ == "__main__":
4494     import argparse
4495
4496     # Parse arguments
4497     parser = argparse.ArgumentParser(description='Run a load test on the RAG system')
4498     parser.add_argument('--workers', type=int, default=3,
4499                         help='Number of concurrent workers')
4500     parser.add_argument('--queries', type=int, default=5,
4501                         help='Number of queries per worker')
4502     args = parser.parse_args()
4503
4504     # Run load test
4505     run_load_test(args.workers, args.queries)

```

#### Checkpoint Questions:

1. What are the key metrics to monitor in a RAG system under load?
2. How does concurrent access impact the performance of different components (vector DB, embedding model, LLM)?
3. What options exist for scaling the system if it becomes performance-limited?
4. **Additional Challenge:** Create a benchmark suite that evaluates the system's accuracy using a set of predefined questions with known answers.



### Lab 4.3: Creating an MLflow Client for the Flask App.

#### Learning Objectives:

- Create a client to interact with the MLflow serving endpoint from Flask
- Implement error handling and logging for MLflow interactions
- Design a clean API abstraction for your web application

#### Exercises:

1. Create the MLflow client for the Flask app:

```
# flask-app/utils/mlflow_client.py
import requests
import json
import logging
from typing import Dict, Any, Optional

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

class MLflowClient:
    def __init__(self, endpoint_url: str):
        """
        Initialize the MLflow client.

        Args:
            endpoint_url: URL of the MLflow serving endpoint
        """
        self.endpoint_url = endpoint_url
        logger.info(f"Initialized MLflow client for endpoint: {endpoint_url}")

    def predict(self, query: str) -> Dict[str, Any]:
        """
        Make a prediction using the MLflow serving endpoint.

        Args:
            query: Query text

        Returns:
            Prediction result
        """
        logger.info(f"Sending query to MLflow endpoint: {query}")

        # Create payload
        payload = {
            "query": query
        }
```

```

4560         # Send request
4561         response = requests.post(
4562             f"{self.endpoint_url}/invocations",
4563             json=payload,
4564             headers={"Content-Type": "application/json"}
4565         )
4566
4567         # Check response
4568         if response.status_code != 200:
4569             logger.error(f"Error from MLflow endpoint: {response.text}")
4570             raise Exception(f"Error from MLflow endpoint: {response.text}")
4571
4572         # Parse response
4573         result = response.json()
4574
4575         logger.info(f"Received response from MLflow endpoint")
4576         return result
4577
4578     def is_alive(self) -> bool:
4579         """
4580         Check if the MLflow endpoint is alive.
4581
4582         Returns:
4583             True if the endpoint is alive, False otherwise
4584         """
4585         try:
4586             response = requests.get(f"{self.endpoint_url}/ping")
4587             return response.status_code == 200
4588         except:
4589             return False
4590
4591     def create_mlflow_client(host: str = "localhost", port: int = 5001) -> MLflowClient:
4592         """
4593         Create an MLflow client.
4594
4595         Args:
4596             host: Host of the MLflow server
4597             port: Port of the MLflow server
4598
4599         Returns:
4600             MLflow client
4601         """
4602         endpoint_url = f"http://{host}:{port}"
4603         return MLflowClient(endpoint_url)

```

2. Create a utility for pipeline triggering:

```
touch flask-app/utils/pipeline_trigger.py
```

3. Add the following code to flask-app/utils/pipeline\_trigger.py:

```

4607 import subprocess
4608 import logging
4609 import threading
4610 from pathlib import Path
4611 import sys
4612
4613 # Configure logging
4614 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
4615 %(message)s')
4616 logger = logging.getLogger(__name__)
4617
4618 def run_pipeline_async(pdf_dir, rebuild=False):
4619     """
4620     Run the pipeline asynchronously in a separate thread.
4621
4622     Args:
4623         pdf_dir: Directory containing PDF files
4624         rebuild: Whether to rebuild the vector index
4625     """
4626     # Get project root
4627     project_root = Path(__file__).resolve().parent.parent.parent
4628
4629     def _run_pipeline():
4630         try:
4631             logger.info(f"Starting pipeline with PDF directory: {pdf_dir}")
4632
4633             # Build command
4634             cmd = [
4635                 sys.executable,
4636                 str(project_root / "app" / "pipeline.py"),
4637                 "--pdf-dir", pdf_dir
4638             ]
4639
4640             if rebuild:
4641                 cmd.append("--rebuild")
4642
4643             # Run pipeline
4644             process = subprocess.Popen(
4645                 cmd,
4646                 stdout=subprocess.PIPE,
4647                 stderr=subprocess.PIPE,
4648                 universal_newlines=True
4649             )
4650
4651             # Get output
4652             stdout, stderr = process.communicate()
4653
4654             if process.returncode != 0:
4655                 logger.error(f"Pipeline failed with return code {process.returncode}")

```

```

4654         logger.error(f"Error output: {stderr}")
4655     else:
4656         logger.info("Pipeline completed successfully")
4657
4658     except Exception as e:
4659         logger.error(f"Error running pipeline: {str(e)}")
4660
4661     # Start thread
4662     thread = threading.Thread(target=_run_pipeline)
4663     thread.daemon = True
4664     thread.start()
4665
4666     return {
4667         'status': 'started',
4668         'message': 'Pipeline started in the background'
4669     }

```

**Professor's Hint:** When triggering long-running processes from a web application, it's best to run them asynchronously to avoid blocking the web server. This keeps the user interface responsive while the processing happens in the background.

## Lab 5: Flask Web Interface

### Lab 5.1: Basic Flask Application Setup.

#### Learning Objectives:

- Set up a Flask web application
- Create a user interface for the RAG system
- Implement file upload and document management

#### Exercises:

1. Create a basic Flask application structure:

```

4684 cd flask-app
4685 touch app.py
4686 touch config.py
4687

```

2. Add the following to config.py:

```

4690 import os
4691 from pathlib import Path
4692
4693 # Base directory
4694 BASE_DIR = Path(__file__).resolve().parent
4695
4696 # Flask settings
4697 SECRET_KEY = "change-this-in-production"
4698 DEBUG = True
4699
4700 # Upload settings

```

```

4701 UPLOAD_FOLDER = os.path.join(Path(__file__).resolve().parent.parent, "data", "pdfs")
4702 MAX_CONTENT_LENGTH = 16 * 1024 * 1024 # 16 MB max upload size
4703 ALLOWED_EXTENSIONS = {'pdf'}
4704
4705 # MLflow settings
4706 MLFLOW_HOST = "localhost"
4707 MLFLOW_PORT = 5001

```

3. Add the following to app.py:

```

4710 import os
4711 from flask import Flask, request, render_template, flash, redirect, url_for, jsonify
4712 from werkzeug.utils import secure_filename
4713 import logging
4714 import sys
4715 from pathlib import Path
4716
4717 # Configure logging
4718 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
4719             %(message)s')
4720 logger = logging.getLogger(__name__)
4721
4722 # Add project root to Python path
4723 sys.path.insert(0, str(Path(__file__).resolve().parent.parent))
4724
4725 # Import configuration
4726 from flask_app.config import SECRET_KEY, DEBUG, UPLOAD_FOLDER, MAX_CONTENT_LENGTH,
4727     ALLOWED_EXTENSIONS
4728 from flask_app.utils.mlflow_client import create_mlflow_client
4729
4730 # Create Flask app
4731 app = Flask(__name__)
4732 app.config['SECRET_KEY'] = SECRET_KEY
4733 app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
4734 app.config['MAX_CONTENT_LENGTH'] = MAX_CONTENT_LENGTH
4735
4736 # Ensure upload folder exists
4737 os.makedirs(UPLOAD_FOLDER, exist_ok=True)
4738
4739 # Create MLflow client
4740 try:
4741     from flask_app.config import MLFLOW_HOST, MLFLOW_PORT
4742     mlflow_client = create_mlflow_client(MLFLOW_HOST, MLFLOW_PORT)
4743     logger.info(f"MLflow client created for endpoint: http://{MLFLOW_HOST}:{MLFLOW_PORT}")
4744 except Exception as e:
4745     logger.error(f"Failed to create MLflow client: {str(e)}")
4746     mlflow_client = None
4747
4748 def allowed_file(filename):
4749     """Check if a file has an allowed extension."""

```

```

4748         return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
4749
4750 @app.route('/')
4751 def index():
4752     """Render the home page."""
4753     return render_template('index.html')
4754
4755 @app.route('/upload', methods=['GET', 'POST'])
4756 def upload():
4757     """Handle file uploads."""
4758     if request.method == 'POST':
4759         # Check if the post request has the file part
4760         if 'file' not in request.files:
4761             flash('No file part', 'error')
4762             return redirect(request.url)
4763
4764         file = request.files['file']
4765
4766         # If user does not select file, browser also
4767         # submit an empty part without filename
4768         if file.filename == '':
4769             flash('No selected file', 'error')
4770             return redirect(request.url)
4771
4772         if file and allowed_file(file.filename):
4773             filename = secure_filename(file.filename)
4774             file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
4775             file.save(file_path)
4776             flash(f'File {filename} uploaded successfully!', 'success')
4777
4778             # Redirect to document list
4779             return redirect(url_for('documents'))
4780         else:
4781             flash('File type not allowed', 'error')
4782             return redirect(request.url)
4783
4784     return render_template('upload.html')
4785
4786 @app.route('/documents')
4787 def documents():
4788     """List uploaded documents."""
4789     # Get list of PDFs in upload folder
4790     pdfs = []
4791     for filename in os.listdir(app.config['UPLOAD_FOLDER']):
4792         if allowed_file(filename):
4793             file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
4794             file_stat = os.stat(file_path)
4795             pdfs.append({
4796                 'filename': filename,
4797                 'size': file_stat.st_size,

```

```

4795         'modified': file_stat.st_mtime
4796     })
4797
4798     # Sort by modified time (newest first)
4799     pdfs = sorted(pdfs, key=lambda x: x['modified'], reverse=True)
4800
4801     return render_template('documents.html', documents=pdfs)
4802
4803 @app.route('/ask', methods=['GET', 'POST'])
4804 def ask():
4805     """Ask a question."""
4806     if request.method == 'POST':
4807         return redirect(url_for('ask'))
4808
4809     return render_template('ask.html')
4810
4811 @app.route('/api/ask', methods=['POST'])
4812 def api_ask():
4813     """API endpoint for asking questions."""
4814     data = request.get_json()
4815
4816     if not data or 'question' not in data:
4817         return jsonify({'error': 'Missing question parameter'}), 400
4818
4819     question = data['question']
4820
4821     if not mlflow_client or not mlflow_client.is_alive():
4822         error_msg = "MLflow endpoint is not available. Please make sure the model is deployed."
4823         logger.error(error_msg)
4824         return jsonify({'error': error_msg}), 503
4825
4826     try:
4827         # Process question
4828         logger.info(f"Processing question: {question}")
4829         response = mlflow_client.predict(question)
4830
4831         return jsonify(response)
4832     except Exception as e:
4833         logger.error(f"Error processing question: {str(e)}")
4834         return jsonify({'error': str(e)}), 500
4835
4836 @app.route('/api/health')
4837 def health():
4838     """Health check endpoint."""
4839     mlflow_status = mlflow_client.is_alive() if mlflow_client else False
4840
4841     return jsonify({
4842         'status': 'ok',
4843         'mlflow': mlflow_status
4844     })

```

```
if __name__ == '__main__':
    app.run(debug=DEBUG, host='0.0.0.0', port=8000)
```

Appendix: Diagrams

Class Diagram. The class diagram is shown below:

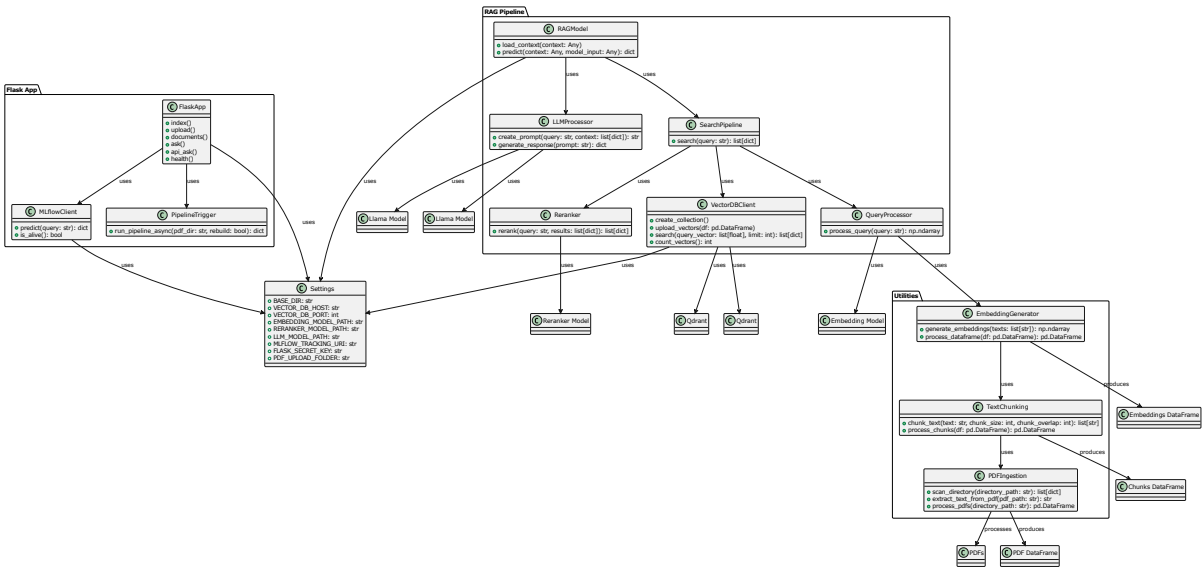


Fig. 1. Class Diagram

Use Case Diagram. The use case diagram is shown below:

Component Diagram. The component diagram is shown below:

Deployment Diagram. The deployment diagram is shown below:

The is it for now :).



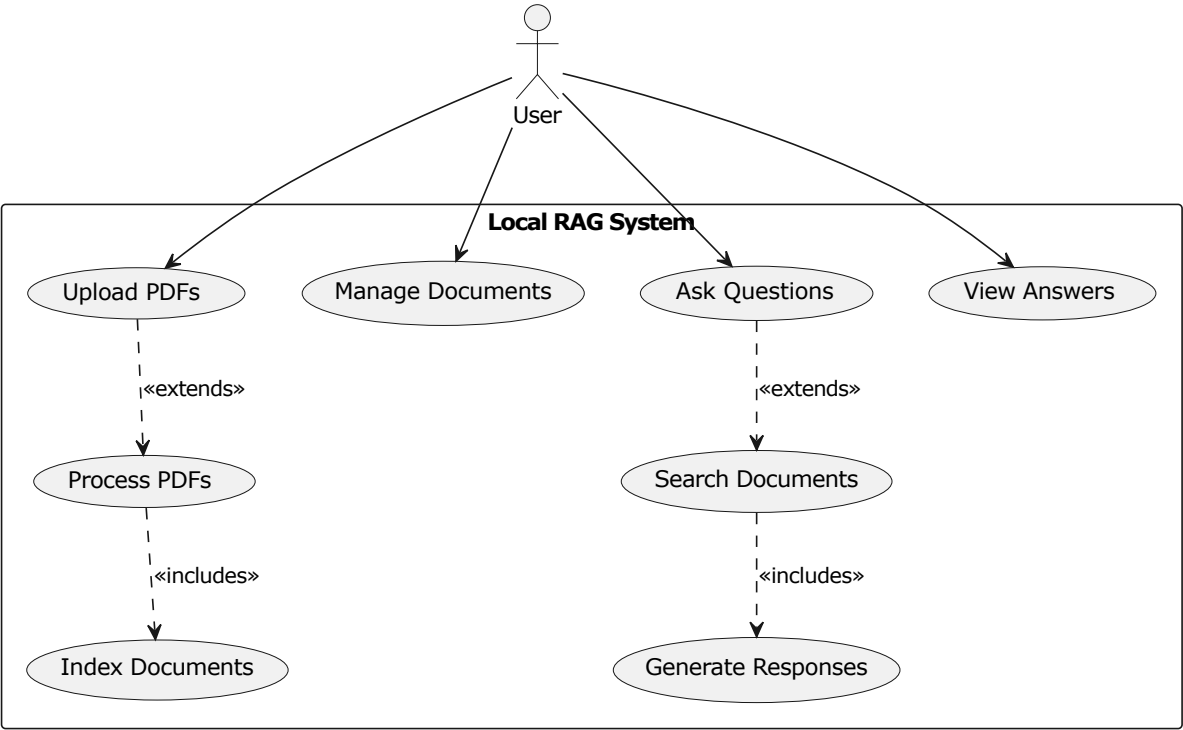


Fig. 2. Use Case Diagram

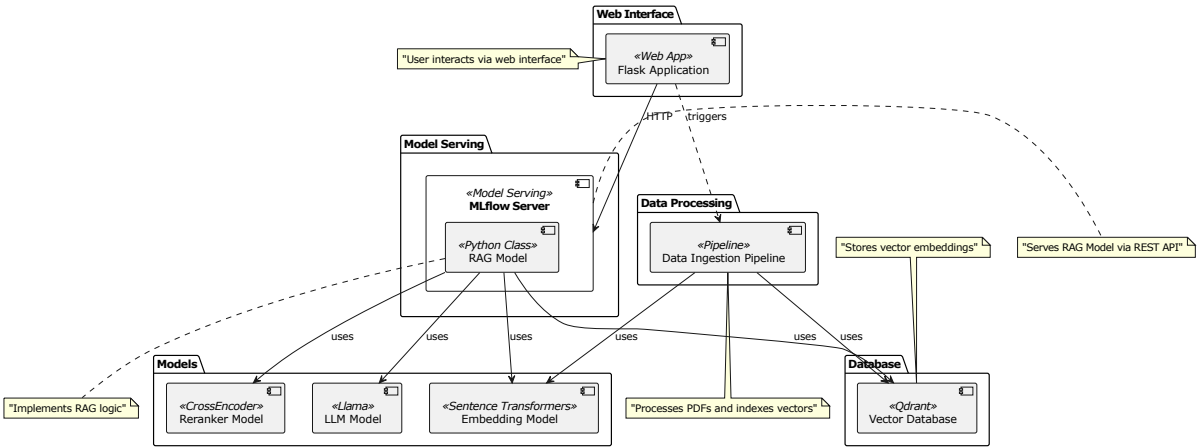


Fig. 3. Component Diagram

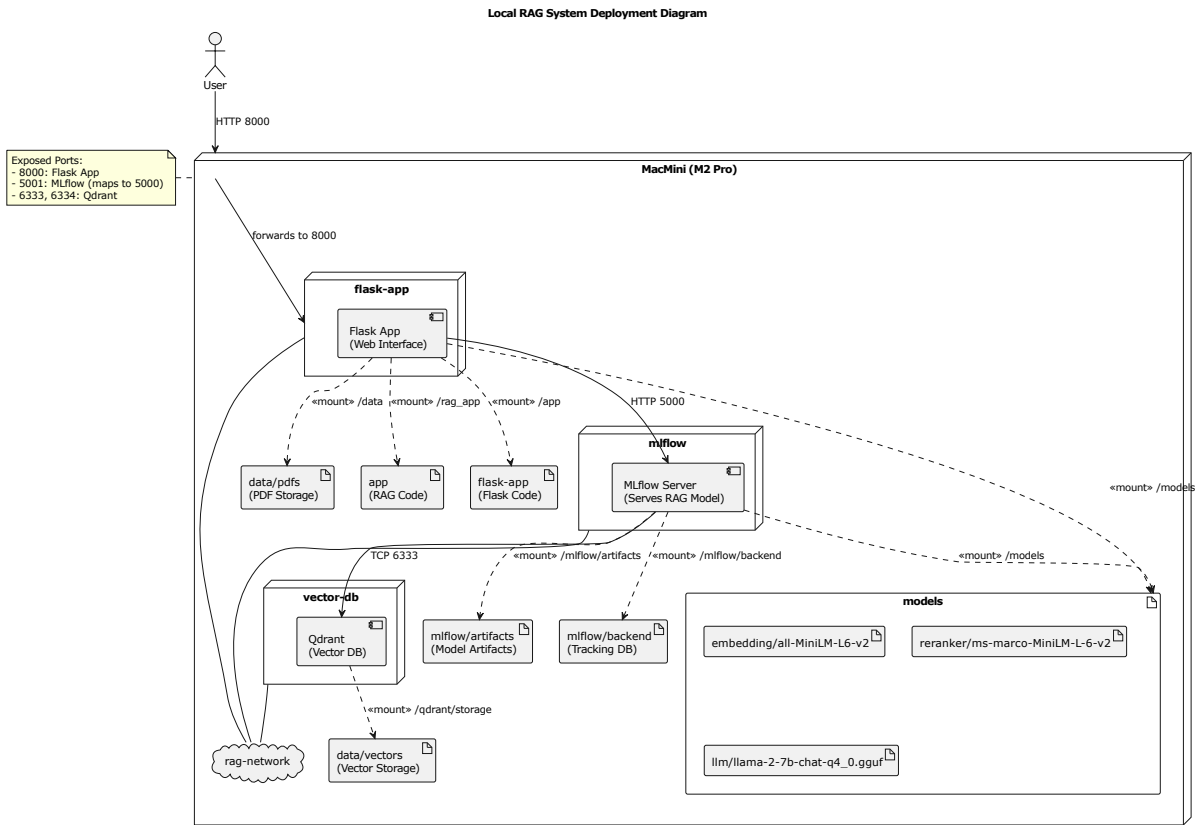


Fig. 4. Deployment Diagram