"""

Main entry point for the RAG pipeline application.

This script orchestrates the entire RAG pipeline process from document

loading to query answering and evaluation.

"""

import argparse

import sys

from pathlib import Path

from typing import List, Optional

import logging

from config.settings import settings

from src.utils.logging\_config import setup\_logging, get\_logger

from src.data\_loader.pdf\_loader import PDFDocumentLoader

from src.text\_processing.chunker import DocumentChunker

from src.embeddings.openai\_embeddings import OpenAIEmbeddingModel

from src.vector\_store.chroma\_store import ChromaVectorStore

from src.llm.openai\_llm import OpenAILLM

from src.retrieval.qa\_chain import QAChain

from src.evaluation.ragas\_evaluator import RagasEvaluator

from src.utils.exceptions import RAGPipelineError

# Set up logging

setup\_logging(

log\_level=settings.log\_level,

log\_file=settings.log\_file,

enable\_console=True

)

logger = get\_logger(\_\_name\_\_)

class RAGPipeline:

"""Main RAG pipeline orchestrator."""

def \_\_init\_\_(self):

"""Initialize the RAG pipeline components."""

logger.info("Initializing RAG Pipeline")

# Initialize components

self.pdf\_loader = PDFDocumentLoader()

self.chunker = DocumentChunker(

chunk\_size=settings.chunk\_size,

chunk\_overlap=settings.chunk\_overlap

)

self.embedding\_model = OpenAIEmbeddingModel(

api\_key=settings.openai\_api\_key,

model\_name=settings.embedding\_model

)

self.vector\_store = ChromaVectorStore(

embedding\_model=self.embedding\_model,

persist\_directory=settings.vector\_store\_persist\_directory,

collection\_name=settings.collection\_name

)

self.llm = OpenAILLM(

api\_key=settings.openai\_api\_key,

model\_name=settings.llm\_model,

temperature=settings.temperature,

max\_tokens=settings.max\_tokens,

top\_p=settings.top\_p,

frequency\_penalty=settings.frequency\_penalty,

presence\_penalty=settings.presence\_penalty

)

self.qa\_chain = None

logger.info("RAG Pipeline initialized successfully")

def load\_documents(self, file\_paths: List[Path]) -> int:

"""

Load documents from file paths.

Args:

file\_paths: List of document file paths

Returns:

Number of documents loaded

"""

logger.info(f"Loading {len(file\_paths)} documents")

# Load PDFs

documents = self.pdf\_loader.load\_multiple(file\_paths)

# Chunk documents

chunks = self.chunker.process(documents)

# Add to vector store

self.vector\_store.add\_documents(chunks)

# Persist vector store

self.vector\_store.persist()

logger.info(f"Successfully loaded and processed {len(chunks)} chunks")

return len(chunks)

def initialize\_qa\_chain(self) -> None:

"""Initialize the QA chain."""

self.qa\_chain = QAChain(

llm=self.llm,

vector\_store=self.vector\_store,

retriever\_kwargs=settings.retriever\_search\_kwargs

)

logger.info("QA chain initialized")

def answer\_question(self, question: str) -> dict:

"""

Answer a question using the RAG pipeline.

Args:

question: User question

Returns:

Response dictionary with answer and sources

"""

if not self.qa\_chain:

self.initialize\_qa\_chain()

return self.qa\_chain.run(question)

def evaluate(self, evaluation\_data: List[dict]) -> dict:

"""

Evaluate the pipeline using Ragas.

Args:

evaluation\_data: List of evaluation samples

Returns:

Evaluation results

"""

if not self.qa\_chain:

self.initialize\_qa\_chain()

evaluator = RagasEvaluator(

qa\_chain=self.qa\_chain,

api\_key=settings.openai\_api\_key

)

results\_df = evaluator.evaluate(evaluation\_data)

metrics = evaluator.get\_metrics()

return {

"results\_dataframe": results\_df,

"metrics": metrics

}

def main():

"""Main function to run the RAG pipeline."""

parser = argparse.ArgumentParser(description="RAG Pipeline")

parser.add\_argument(

"--mode",

choices=["index", "query", "evaluate", "demo"],

required=True,

help="Mode to run the pipeline in"

)

parser.add\_argument(

"--documents",

nargs="+",

help="Document files to index (for index mode)"

)

parser.add\_argument(

"--question",

help="Question to ask (for query mode)"

)

parser.add\_argument(

"--evaluation-file",

help="Path to evaluation data file (for evaluate mode)"

)

parser.add\_argument(

"--load-existing",

action="store\_true",

help="Load existing vector store instead of creating new"

)

args = parser.parse\_args()

try:

# Initialize pipeline

pipeline = RAGPipeline()

if args.mode == "index":

# Index documents

if not args.documents:

logger.error("No documents provided for indexing")

sys.exit(1)

file\_paths = [Path(doc) for doc in args.documents]

num\_chunks = pipeline.load\_documents(file\_paths)

print(f"Successfully indexed {num\_chunks} chunks from {len(file\_paths)} documents")

elif args.mode == "query":

# Load existing vector store if requested

if args.load\_existing:

pipeline.vector\_store.load()

# Answer question

if not args.question:

logger.error("No question provided")

sys.exit(1)

response = pipeline.answer\_question(args.question)

print(f"\nQuestion: {response['query']}")

print(f"\nAnswer: {response['answer']}")

print(f"\nSources: {len(response['source\_documents'])} documents")

for i, doc in enumerate(response['source\_documents']):

print(f"\nSource {i+1}:")

print(f"Content: {doc.page\_content[:200]}...")

print(f"Metadata: {doc.metadata}")

elif args.mode == "evaluate":

# Run evaluation

if args.load\_existing:

pipeline.vector\_store.load()

# Load evaluation data (implement based on your needs)

import json

if args.evaluation\_file:

with open(args.evaluation\_file, 'r') as f:

evaluation\_data = json.load(f)

else:

# Use default evaluation data

evaluation\_data = [] # Add your default evaluation data

results = pipeline.evaluate(evaluation\_data)

print(f"\nEvaluation Results:")

print(f"Metrics: {results['metrics']}")

elif args.mode == "demo":

# Run a demo with sample documents and questions

print("Running demo mode...")

# Index sample documents

sample\_docs = list(settings.documents\_directory.glob("\*.pdf"))

if sample\_docs:

pipeline.load\_documents(sample\_docs)

# Ask sample questions

sample\_questions = [

"How do you install the SIM card in the iPhone?",

"What is the purpose of the Ring/Silent Switch?",

"How do you set up a portable Wi-Fi hotspot on Android?"

]

for question in sample\_questions:

response = pipeline.answer\_question(question)

print(f"\nQ: {question}")

print(f"A: {response['answer'][:300]}...")

else:

print("No sample documents found in", settings.documents\_directory)

except RAGPipelineError as e:

logger.error(f"Pipeline error: {str(e)}")

sys.exit(1)

except Exception as e:

logger.error(f"Unexpected error: {str(e)}", exc\_info=True)

sys.exit(1)

if \_\_name\_\_ == "\_\_main\_\_":

main()