**Document Retrieval System Implementation**

**Overview**

Your task is to implement a document retrieval system that helps researchers find relevant information from academic papers. You'll build the core components of a retrieval-augmented architecture, focusing on document processing, embedding generation, and semantic search capabilities.

**System Requirements**

**Environment Setup**

* Python 3.8+
* Dependencies will be listed in requirements.txt
* Local development environment (no cloud services required)

**Core Features to Implement**

1. Document Processing
   * Parse PDF and TXT files
   * Split documents into chunks with configurable overlap
   * Generate text embeddings using basic techniques (TF-IDF or word2vec)
2. Vector Storage
   * Implement in-memory vector storage
   * Support adding and querying embeddings
   * Include metadata management
3. Query Processing
   * Process user queries
   * Retrieve relevant document chunks
   * Rank results by relevance
   * Handle error cases gracefully
4. Mock LLM Integration
   * Implement response generation simulation
   * Handle rate limiting
   * Manage API-like error scenarios

**Provided Resources**

**1. Starting Code Template**

python

from typing import List, Dict

import numpy as np

class Document:

def \_\_init\_\_(self, file\_path: str):

self.file\_path = file\_path

self.content = None

self.chunks = None

self.embeddings = None

*# TODO: Implement methods*

class VectorStore:

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

self.embeddings = []

self.metadata = []

*# TODO: Implement methods*

class MockLLM:

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

self.request\_count = 0

self.rate\_limit = 10 *# requests per minute*

*# TODO: Implement methods*

*# Add your implementation here*

**Implementation Requirements**

**1. Document Class**

python

def load\_document(self) -> str:

"""

Load document content from file.

Handle both PDF and TXT formats.

"""

pass

def chunk\_text(self, chunk\_size: int = 500, overlap: int = 50) -> List[str]:

"""

Split document into overlapping chunks.

"""

pass

def generate\_embeddings(self, chunks: List[str]) -> List[np.ndarray]:

"""

Create embeddings for text chunks.

Use either TF-IDF or word2vec approach.

"""

pass

**2. VectorStore Class**

python

def add\_embeddings(self, embeddings: List[np.ndarray], metadata: List[dict]):

"""

Store embeddings with their metadata.

"""

pass

def search(self, query\_embedding: np.ndarray, k: int = 5) -> List[dict]:

"""

Find most similar chunks using cosine similarity.

Return chunks with their metadata.

"""

pass

**3. MockLLM Class**

python

def generate\_response(self, prompt: str, context: List[str]) -> dict:

"""

Simulate LLM response generation.

Include rate limiting and error simulation.

"""

pass

**Evaluation Criteria**

**1. Code Quality (30%)**

* Clean, well-organized code
* Proper documentation
* Error handling
* Type hints

**2. System Design (30%)**

* Class structure
* Data flow
* Memory efficiency
* Error scenarios

**3. RAG Implementation (40%)**

* Chunking strategy
* Embedding generation
* Search effectiveness
* Result ranking

**Submission Requirements**

1. Code Files
   * Implementation of all required classes
   * requirements.txt
2. Documentation
   * Setup instructions
   * Usage examples
   * Design decisions explanation
   * Limitations and potential improvements
3. Results
   * Performance metrics
   * Error handling examples

**Example Usage**

Python

def main():

*# Initialize system*

doc\_processor = Document("data/paper1.pdf")

vector\_store = VectorStore()

mock\_llm = MockLLM()

*# Process document*

content = doc\_processor.load\_document()

chunks = doc\_processor.chunk\_text()

embeddings = doc\_processor.generate\_embeddings(chunks)

vector\_store.add\_embeddings(embeddings,

metadata=[{"source": "paper1.pdf"}] \* len(chunks))

*# Process query*

query = "What are the main impacts of climate change on agriculture?"

try:

*# Your query processing implementation*

response = process\_query(query, vector\_store, mock\_llm)

print(f"Response: {response}")

except Exception as e:

print(f"Error: {e}")

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

main()

**Bonus Points**

* Caching implementation
* Alternative embedding methods
* API interface
* Performance optimizations
* Additional error handling