AI/ML Developer Coding Challenge

Intelligent Document Understanding API

Overview

In this challenge, you will develop an end-to-end document processing API that extracts structured information from unstructured documents. The system will combine OCR technology, vector database retrieval, and large language models to deliver a complete document understanding solution.

Problem Statement

You are tasked with building an API that can:

1. Accept document uploads (images or PDFs)
2. Extract text content using Optical Character Recognition (OCR)
3. Identify document type via semantic search against a vector database
4. Extract structured data fields using Large Language Models (LLMs)
5. Return a standardized JSON response containing all extracted information

Dataset Resources

Use documents from the following collection for development and testing:

- **Real-World Documents Collection** (Kaggle Link)

Technical Requirements

1. Optical Character Recognition

- Implement OCR processing for both images and PDF documents
- Utilize industry-standard OCR tools (Tesseract, EasyOCR, or equivalent)
- Optimize for text extraction quality, handling various document layouts

2. Vector Database Implementation

- Process and index sample documents:
  - Apply OCR to extract text from each document
  - Generate text embeddings using a suitable embedding model
  - Store embeddings and document type metadata in a vector database (FAISS, Chroma, etc.)
  - Implement efficient similarity search capabilities

3. API Development with FastAPI

- Create a robust API using FastAPI framework
- Implement a `/extract_entities/` POST endpoint that:
  - Accepts multipart form data with document file uploads
  - Validates input file formats
  - Processes documents asynchronously where appropriate
  - Returns structured JSON responses

4. Document Classification

- Create a document type detection pipeline:
  - Apply OCR to uploaded documents
  - Generate embeddings for extracted text
  - Perform semantic search against the indexed vector database
  - Return the closest matching document type with confidence score

5. Entity Extraction with LLMs

- Implement prompt-based entity extraction using an LLM:
  - Select an appropriate model (OpenAI GPT-3.5, Llama-2, or equivalent local model)
  - Develop document type-specific prompts for entity extraction
  - Example prompt template:

```
Given the following text extracted from a document of type '{document_type}',
extract these fields: {field_list}.
Return your response as a valid JSON object with no additional text.

Document Text:
{document_text}
```

  - Parse and validate LLM responses into structured JSON

6. API Response Format

The API should return standardized JSON responses:

```json
{
  "document_type": "Invoice",
  "confidence": 0.92,
  "entities": {
    "invoice_number": "INV-12345",
    "date": "2024-01-01",
    "total_amount": "$450.00",
    "vendor_name": "ABC Corp"
  },
  "processing_time": "1.25s"
}
```

Submission Requirements

- Complete Python codebase organized into logical modules
- Comprehensive `requirements.txt` file listing all dependencies
- Example requests and responses demonstrating API functionality
- README with:
  - Project overview
  - Installation instructions
  - Configuration guidelines
  - API documentation
  - Testing procedures

Evaluation Criteria

Your submission will be evaluated based on:

- **Code Quality**: Well-structured, documented, and maintainable code
- **OCR Performance**: Accurate text extraction from diverse document formats
- **Vector Search Implementation**: Effective document type classification
- **Entity Extraction**: Precision and completeness of extracted fields
- **Prompt Engineering**: Thoughtful prompt design for optimal LLM performance
- **Error Handling**: Graceful handling of edge cases and failures
- **Documentation**: Clear and comprehensive project documentation

Bonus Points

- Handling low-quality OCR text with additional preprocessing
- Implementing confidence scoring for extracted entities
- Adding unit and integration tests
- Containerizing the application with Docker
- Including a simple web interface for testing

Good luck!