M&A Due Diligence Analyzer - Setup Guide

Quick Start (5 Minutes)

1. Create Project Structure

```bash
mkdir ma-diligence
cd ma-diligence

# Create directories
mkdir -p api analyzers parsers models utils tests/test_documents

# Create __init__.py files
touch api/__init__.py analyzers/__init__.py parsers/__init__.py models/__init__.py utils/__init__.py
```

2. Install Dependencies

```bash
# Create virtual environment
python -m venv venv

# Activate it
# On Linux/Mac:
source venv/bin/activate
# On Windows:
# venv\Scripts\activate

# Install packages
pip install fastapi uvicorn[standard] python-multipart pymupdf anthropic openai pydantic pydantic-settings python-dotenv aio
```

3. Configure API Keys

Create a `.env` file in the project root:

```bash
ANTHROPIC_API_KEY=sk-ant-your-key-here
OPENAI_API_KEY=sk-your-key-here
```

Get your API keys:

- Anthropic: https://console.anthropic.com/
- OpenAI: https://platform.openai.com/api-keys

4. Copy the Code

Copy all the code files I provided into their respective directories:

- `utils/config.py`
- `models/schemas.py`
- `parsers/pdf_parser.py`
- `analyzers/rule_engine.py`
- `analyzers/llm_analyzer.py`
- `analyzers/aggregator.py`
- `api/main.py`
- `test_analyzer.py` (in project root)

5. Test It

Option A: Test with standalone script

```bash
# Put your test PDF in tests/test_documents/
python test_analyzer.py tests/test_documents/your_contract.pdf
```

Option B: Run the API

```bash
# Start the server
uvicorn api.main:app --reload

# In another terminal, test with curl:
curl -X POST "http://localhost:8000/analyze" \
  -F "file=@tests/test_documents/your_contract.pdf"
```

Detailed Setup

Project Structure

```
ma-diligence/
├── api/
│   ├── __init__.py
│   └── main.py              # FastAPI app
├── analyzers/
│   ├── __init__.py
│   ├── rule_engine.py       # Rule-based checks
│   ├── llm_analyzer.py      # Claude + GPT analysis
│   └── aggregator.py        # Result aggregation
├── parsers/
│   ├── __init__.py
│   └── pdf_parser.py        # PDF text extraction
├── models/
│   ├── __init__.py
│   └── schemas.py           # Pydantic models
├── utils/
│   ├── __init__.py
│   └── config.py            # Configuration
├── tests/
│   └── test_documents/      # Your test PDFs
├── .env                     # API keys (don't commit!)
├── .gitignore
├── requirements.txt
├── test_analyzer.py         # Standalone test script
└── README.md
```

Dependencies Explained

- **fastapi**: Modern web framework for building APIs
- **uvicorn**: ASGI server to run FastAPI
- **pymupdf**: PDF parsing library (fast and reliable)
- **anthropic**: Claude API client
- **openai**: GPT API client
- **pydantic**: Data validation and settings management
- **aiohttp**: Async HTTP client (for parallel requests)

Configuration Options

Edit `utils/config.py` to adjust:

```python
# Model selection
claude_model = "claude-sonnet-4-5-20250929"  # or use Haiku for cheaper
gpt_model = "gpt-4o"  # or gpt-4o-mini

# Analysis parameters
max_tokens = 4096
temperature = 0.1  # Low = more consistent

# Performance
enable_parallel_llm = True  # Run Claude + GPT simultaneously
```

Usage Examples

1. Standalone Testing (No API)

```python
python test_analyzer.py path/to/contract.pdf
```

This will:

1. Parse the PDF
2. Run rule-based checks
3. Run Claude + GPT analysis
4. Print a detailed report

2. API Usage

Start the server:

```bash
uvicorn api.main:app --reload --host 0.0.0.0 --port 8000
```

Test with curl:

```bash
# Basic analysis
curl -X POST http://localhost:8000/analyze \
  -F "file=@contract.pdf"

# Disable GPT (use only Claude + rules)
curl -X POST "http://localhost:8000/analyze?use_gpt=false" \
  -F "file=@contract.pdf"

# Rules only (fast, cheap)
curl -X POST "http://localhost:8000/analyze?use_claude=false&use_gpt=false" \
  -F "file=@contract.pdf"
```

Test with Python:

```python
import requests

url = "http://localhost:8000/analyze"
files = {"file": open("contract.pdf", "rb")}
response = requests.post(url, files=files)

result = response.json()
print(f"Found {result['total_flags']} red flags")
print(f"Risk score: {result['overall_risk_score']}/10")

for flag in result['flags'][:5]:
    print(f"\n[{flag['severity']}] {flag['title']}")
    print(f"  {flag['description']}")
```

3. Batch Analysis

```bash
curl -X POST http://localhost:8000/analyze/batch \
  -F "files=@contract1.pdf" \
  -F "files=@contract2.pdf" \
  -F "files=@contract3.pdf"
```

Cost Optimization

Strategy 1: Tiered Analysis

Run cheap checks first, expensive LLMs only on flagged sections:

```python
# In llm_analyzer.py, modify analyze() to:
# 1. Only send flagged sections to LLMs
# 2. Use Sonnet for initial pass, Opus only for unclear cases
```

Strategy 2: Model Selection

Model	Cost	Speed	Quality	Use Case
Claude Haiku	$	Fast	Good	Initial screening
Claude Sonnet	$$	Medium	Excellent	Main analysis
Claude Opus	$$$$	Slow	Best	Complex edge cases
GPT-4o	$$	Fast	Excellent	Redundancy check
GPT-4o-mini	$	Very Fast	Good	Cost-sensitive

Recommended setup:

- Rules: Always (free)
- Sonnet: Main analysis
- GPT-4o: Validation on high-severity findings only

Strategy 3: Caching

Add Redis caching for repeat sections:

```python
# If you see the same clause structure, skip re-analysis
cache_key = hashlib.md5(section_text.encode()).hexdigest()
if cache_key in redis_client:
    return cached_result
```

Troubleshooting

Issue: "Module not found"

Solution: Make sure you're in the project root and virtual environment is activated:

```bash
cd ma-diligence
source venv/
```