Repository Analysis Report

Generated: 2025-07-26 16:35:54

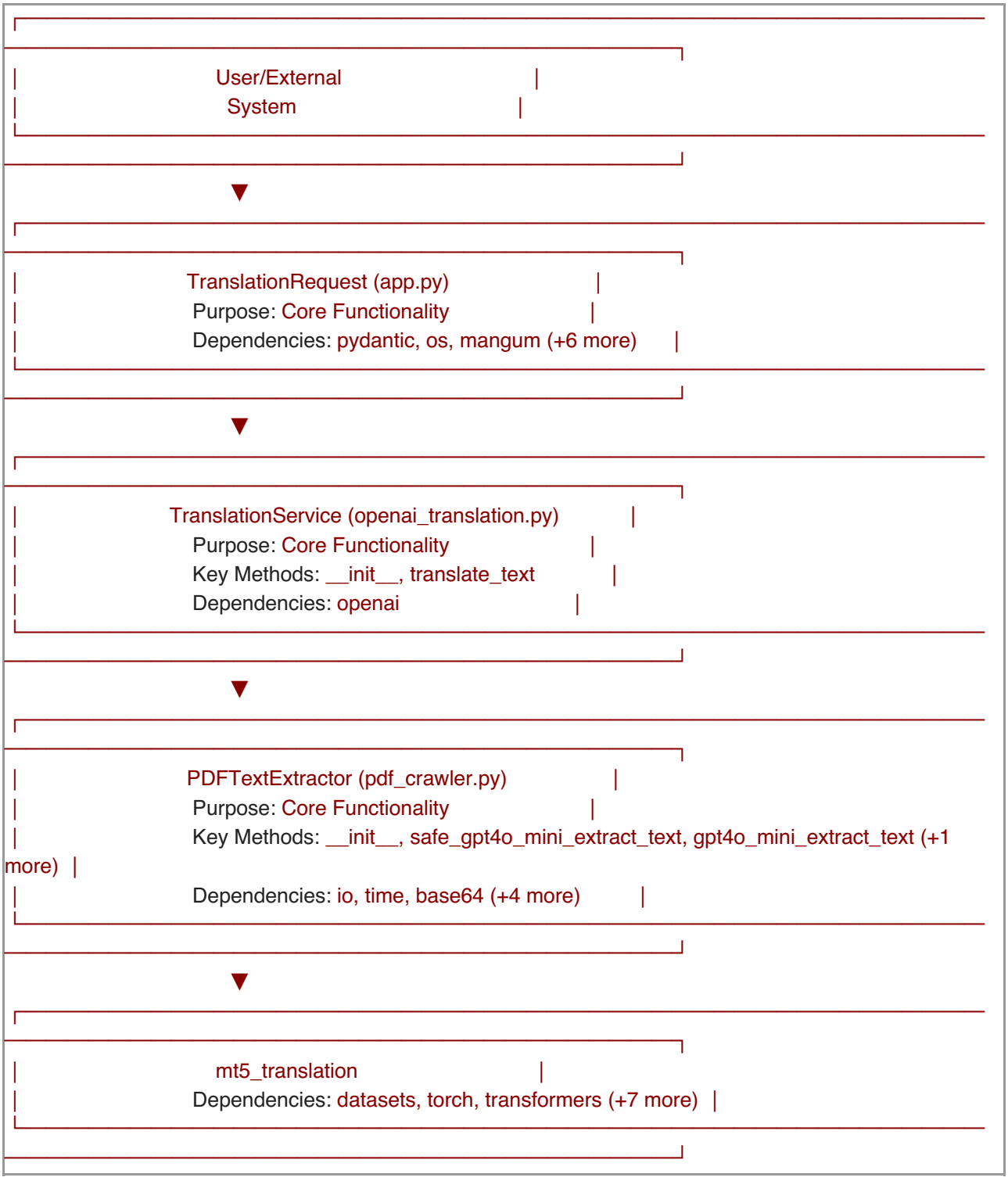
PDF-Translation - Developer Documentation

PDF-Translation - Repository Overview & Workflow

High-Level Architecture

The following diagram shows the main workflow and data flow patterns in this repository:

PDF-Translation - System Architecture



This diagram represents the architecture of the PDF-Translation system. It starts with the user or external system interacting with the TranslationRequest class in app.py. This class then interacts with the TranslationService class in openai_translation.py, which in turn interacts with the PDFTextExtractor class in pdf_crawler.py. Finally, the PDFTextExtractor class interacts with the mt5_translation. The dependencies of each class are also listed in the diagram.

Repository Statistics

Metric	Count
Classes	3

⚙ **Functions**	0
Entry Points	0
Data Flow Steps	0

This overview was automatically generated by analyzing the repository structure, function definitions, and import relationships.

Table of Contents

> **Note for New Developers**: This documentation includes realistic input/output examples throughout. Look for code blocks marked with 'Expected output:' to understand what you should see when running the code.

Getting Started

Prerequisites

Languages Used: Python

Installation

```
git clone <repository-url>
cd PDF-Translation
# Follow language-specific setup instructions
```

API Reference with Input/Output Examples

> **For Beginners**: Each function below includes realistic examples showing exactly what input to provide and what output to expect.

__init__.py

Language: Python

The provided code does not contain any functions or methods. It's a simple Python module initialization file (**init**.py) that imports a class PDFTextExtractor from the pdf_crawler module and includes it in the __all__ variable.

However, I can provide a brief documentation for the PDFTextExtractor class assuming it's a class that extracts text from PDF files.

PDFTextExtractor(file_path)

What it does: This class is used to extract text from a PDF file.

Input:
• file_path (str) - The path to the PDF file from which text needs to be extracted.

Output: An instance of the class PDFTextExtractor.

Quick Example:

```
# Basic usage
extractor = PDFTextExtractor("example.pdf")
text = extractor.get_text()
print(text)  # Output: Text content of the PDF file

# Error handling
try:
    extractor = PDFTextExtractor("invalid.pdf")
except FileNotFoundError as e:
    print(f"Handle error: {e}")
```

Usage Patterns:
• **When to use:** When you need to extract text from a PDF file.
• **Common workflow:** load_file() → PDFTextExtractor() → get_text()
• **Combine with:** save_text_to_file()
• **Avoid:** Trying to extract text from a non-PDF file.

Real-World Example:

```
# Typical production usage
file_path = get_file_path()
extractor = PDFTextExtractor(file_path)
text = extractor.get_text()
save_text_to_file(text, "output.txt")
```

Troubleshooting:
• **Error "FileNotFoundError":** Check if the file path is correct and the file exists.
• **Slow performance:** This could be due to the size of the PDF file. Consider splitting the file.
• **Memory issues:** If the PDF file is too large, it might consume a lot of memory. Consider processing the file in chunks.

---

pdf_crawler.py

Language: Python

init(openai_api_key)

What it does: Initializes the PDFTextExtractor class with the OpenAI API key.

Input:
• openai_api_key (str) - The API key for OpenAI.

Output: None

Quick Example:

```
# Basic usage
pdf_extractor = PDFTextExtractor("your_openai_api_key")
```

Usage Patterns:
• **When to use:** Use this method when you want to instantiate the PDFTextExtractor class.
• **Common workflow:** __init__() → extract_text_from_pdf()
• **Combine with:** None
• **Avoid:** Do not share your OpenAI API key publicly.

Real-World Example:

```
# Typical production usage
pdf_extractor = PDFTextExtractor("your_openai_api_key")
text = pdf_extractor.extract_text_from_pdf("path_to_your_pdf")
```

Troubleshooting:
• **Error "Invalid API key":** Check your OpenAI API key and ensure it's correct.

safe_gpt4o_mini_extract_text(image, retries, sleep_seconds)

What it does: Extracts text from an image using the OpenAI API with retry logic for rate limit errors.

Input:
• image (Image.Image) - The image from which to extract text.
• retries (int, optional) - The number of times to retry if a rate limit error occurs. Default is 5.
• sleep_seconds (int, optional) - The number of seconds to sleep between retries. Default is 10.

Output: str - The extracted text from the image.

Quick Example:

```
# Basic usage
from PIL import Image
img = Image.open("path_to_your_image.jpg")
text = pdf_extractor.safe_gpt4o_mini_extract_text(img)
print(text)  # Output: Extracted text from the image

# Error handling
try:
    text = pdf_extractor.safe_gpt4o_mini_extract_text("invalid_input")
except Exception as e:
    print(f"Handle error: {e}")
```

Usage Patterns:
• **When to use:** Use this method when you want to extract text from an image with retry logic for rate limit errors.

• **Common workflow:** __init__() → safe_gpt4o_mini_extract_text()
• **Combine with:** extract_text_from_pdf()
• **Avoid:** Do not use this method with non-image inputs.

Real-World Example:

```
# Typical production usage
pdf_extractor = PDFTextExtractor("your_openai_api_key")
img = Image.open("path_to_your_image.jpg")
text = pdf_extractor.safe_gpt4o_mini_extract_text(img)
```

Troubleshooting:
• **Error "Invalid image":** Check your image and ensure it's a valid image file.
• **Slow performance:** If the method is slow, try reducing the number of retries or the sleep time between retries.
• **Memory issues:** If you're dealing with large images, consider resizing them before passing to this method.

extract_text_from_pdf(pdf_path)

What it does: Extracts text from a PDF file including text within images.

Input:
• pdf_path (str) - The path to the PDF file.

Output: List[Dict] - A list of dictionaries, each containing the page number, extracted text, and OCR text from images.

Quick Example:

```
# Basic usage
text = pdf_extractor.extract_text_from_pdf("path_to_your_pdf")
print(text)  # Output: List of dictionaries with extracted text

# Error handling
try:
    text = pdf_extractor.extract_text_from_pdf("invalid_path")
except Exception as e:
    print(f"Handle error: {e}")
```

Usage Patterns:
• **When to use:** Use this method when you want to extract text from a PDF file.
• **Common workflow:** __init__() → extract_text_from_pdf()
• **Combine with:** safe_gpt4o_mini_extract_text()
• **Avoid:** Do not use this method with non-PDF inputs.

Real-World Example:

```
# Typical production usage
pdf_extractor = PDFTextExtractor("your_openai_api_key")
text = pdf_extractor.extract_text_from_pdf("path_to_your_pdf")
```

Troubleshooting:
• **Error "Invalid PDF file":** Check your PDF file and ensure it's a valid PDF file.
• **Slow performance:** If the method is slow, consider using a PDF file with fewer pages or images.

• **Memory issues:** If you're dealing with large PDF files, consider splitting them into smaller files before passing to this method.

Architecture Overview

Architecture Analysis: __init__.py

System Architecture Overview

What This Component Does (In Simple Terms):

- Primary responsibility: This code imports the PDFTextExtractor class from the pdf_crawler module.
- Role in the system: It makes the PDFTextExtractor class available for other modules in the package to import.
- Why it exists: It simplifies the import process by allowing other modules to import PDFTextExtractor directly from the package, rather than having to import it from the pdf_crawler module.

Core Components:

- **PDFTextExtractor** (pdf_crawler.py) - This class is responsible for extracting text from PDF files. A typical input would be a path to a PDF file, and the output would be the extracted text.

Data Flow with Examples:
This code doesn't process data, but it does facilitate data flow by making the PDFTextExtractor class available for import. Here's an example of how it might be used:

```
from package_name import PDFTextExtractor

extractor = PDFTextExtractor()
text = extractor.extract_text("path/to/file.pdf")
```

Visual Data Flow:

```
[PDF File] → [PDFTextExtractor] → [Extracted Text]
    ↑            ↓            ↓
"path/to/file.pdf"  "extract_text method"   "extracted text"
```

Design Patterns Used (Simplified):

- **Facade Pattern**:
    - **What it is**: The facade pattern provides a simplified interface to a complex subsystem.
    - **Why it's used here**: It simplifies the import process by allowing PDFTextExtractor to be imported directly from the package.
    - **Example**: from package_name import PDFTextExtractor
    - **Alternative approaches**: Without this, developers would have to import PDFTextExtractor from pdf_crawler, which is less intuitive and requires more knowledge of the package's internal structure.

Key Dependencies Explained:

- **Internal Module** (pdf_crawler):

    - **What it does**: Contains the PDFTextExtractor class, which extracts text from PDF files.
    - **How this code uses it**: This code imports PDFTextExtractor from pdf_crawler.
    - **Data exchange**: When PDFTextExtractor is used, it takes a PDF file as input and returns the extracted text.

Integration Points with Examples:
This code doesn't interact with APIs, databases, or files, and it doesn't handle errors. It simply makes PDFTextExtractor available for import.

Performance & Scalability:
This code doesn't have performance or scalability considerations because it doesn't process data. However, the performance and scalability of PDFTextExtractor would depend on the size and complexity of the PDF files it's used to process.

Improvement Suggestions:
There are no obvious improvements to suggest for this code. It's simple and does its job effectively.

For New Developers:

- **Where to start**: Look at the pdf_crawler.py file to understand how PDFTextExtractor works.
- **Common modifications**: You might add more classes to the __all__ list if you add more classes to the pdf_crawler module.
- **Testing approach**: Test that PDFTextExtractor can be imported directly from the package.
- **Documentation to read**: Look at the documentation for PDFTextExtractor to understand how to use it.

Architecture Analysis: pdf_crawler.py

System Architecture Overview

What This Component Does (In Simple Terms):

- Primary responsibility: This Python script extracts text from PDF files, including text embedded in images within the PDF.
- Role in the system: It's a standalone utility that can be used as part of a larger system for processing and analyzing PDF documents.
- Why it exists: It solves the problem of extracting text from PDFs, including OCR (Optical Character Recognition) on images, which is a common requirement in data extraction and analysis tasks.

Core Components:

- **PDFTextExtractor Class** (pdf_crawler.py) - This class handles the entire process of text extraction from a PDF. It uses the OpenAI API for OCR and the fitz library for PDF processing.
    - Typical input: An API key for OpenAI and a path to a PDF file.
    - Typical output: A list of dictionaries, each containing the page number, extracted text, and OCR text from images for each page in the PDF.

Data Flow with Examples:

```
Input Example: "sample.pdf" (a PDF file with text and images)
    ↓
Step 1: Open the PDF
    → Intermediate result: A `fitz.Document` object representing the PDF
    ↓
Step 2: Extract text and images from each page
    → Intermediate result: A list of dictionaries with page numbers, extracted text, and images
    ↓
Step 3: Perform OCR on images
    → Intermediate result: A list of dictionaries with page numbers, extracted text, and OCR text from images
    ↓
Final Output: A list of dictionaries with page numbers, extracted text, and OCR text from images
```

Visual Data Flow:

```
[PDF File] → [PDFTextExtractor.extract_text_from_pdf] → [PDFTextExtractor.gpt4o_mini_extract_text] → [Output List]
    ↑                ↓                          ↓                    ↓
"sample.pdf"  "fitz.Document object"   "Image object"    "List of dictionaries"
```

Design Patterns Used (Simplified):

- **Retry Pattern**:
    - **What it is**: This pattern involves retrying a failed operation a certain number of times before giving up.
    - **Why it's used here**: It's used to handle rate limit errors from the OpenAI API. If a rate limit error occurs, the script waits for a while and then tries again.
    - **Example**: The safe_gpt4o_mini_extract_text method implements this pattern.
    - **Alternative approaches**: An alternative could be to use a backoff algorithm that increases the wait time between retries.

Key Dependencies Explained:

- **External Library** (openai):
    - **Purpose**: This library provides a Python interface to the OpenAI API, which is used for OCR.
    - **Input it expects**: An image in base64 format.
    - **Output it returns**: The text extracted from the image.
    - **Example usage**: openai.ChatCompletion.create(model="gpt-4o-mini", messages=[...]) → {'choices': [{'message': {'content': 'extracted text'}}]}
- **External Library** (fitz):
    - **Purpose**: This library provides a Python interface to the PDF processing library PyMuPDF.
    - **Input it expects**: A path to a PDF file.
    - **Output it returns**: A fitz.Document object representing the PDF.
    - **Example usage**: fitz.open("sample.pdf") → fitz.Document object

Integration Points with Examples:

API Connections:

- **Service**: OpenAI API
- **Purpose**: To perform OCR on images
- **Request example**:

```
response = openai.ChatCompletion.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": [
                {"type": "text", "text": "If the image doesn't have text within it then you may skip the image and return empty string. Else you have extract the text from the image!"},
                {"type": "image_url", "image_url": {"url": f"{encoded_image}"}}
            ]
        }
    ],
)
# Expected response: {'choices': [{'message': {'content': 'extracted text'}}]}
```

File Operations:

- **File type**: PDF
- **Operation**: Read/Process
- **Example**:

```
# Reading
doc = fitz.open("sample.pdf")
# Expected data format: fitz.Document object
```

Error Handling Patterns:

- **Common errors**: Rate limit errors from the OpenAI API, errors opening images
- **How they're handled**: Rate limit errors are handled with a retry pattern. Errors opening images are caught and ignored, allowing the script to continue processing the rest of the images.
- **Example error flow**: If a rate limit error occurs during OCR, the script waits for 10 seconds and then tries again, up to a maximum of 5 attempts.

Performance & Scalability:

Current Performance Characteristics:

- **Processing speed**: Depends on the size and complexity of the PDF, and the rate limits of the OpenAI API.
- **Memory usage**: Depends on the size of the PDF and the images within it.
- **Bottlenecks**: The OCR process can be slow due to rate limits and the time it takes to process images.

Scaling Considerations:

- **Handle more data**: The script could handle larger PDFs, but it would take longer and use more memory.
- **Concurrent usage**: The script is not currently designed for concurrent usage. Multiple instances could run at the same time, but they would each need their own API key to avoid rate limit issues.
- **Resource limits**: The main resource limits are the memory needed to store the PDF and images,

and the rate limits of the OpenAI API.

Improvement Suggestions:

- **Quick wins**: The script could be optimized to process images in parallel, using Python's multiprocessing or threading libraries.
- **Long-term**: For better scalability, the script could be refactored to process PDFs in chunks, reducing memory usage for large PDFs.
- **Monitoring**: Monitor the time taken for each step of the process, the rate of rate limit errors, and the memory usage.

For New Developers:

- **Where to start**: Start with the PDFTextExtractor class and its extract_text_from_pdf method.
- **Common modifications**: Developers might want to modify the OCR process, for example to use a different OCR API or to improve the image processing.
- **Testing approach**: Test the script with a variety of PDFs, including PDFs with different types of images and different amounts of text.
- **Documentation to read**: The documentation for the openai and fitz libraries, and the OpenAI API documentation.

Class Documentation with Real Examples

> **New Developer Tip**: Each class section includes step-by-step usage examples and common scenarios you can copy and adapt.

Classes in __init__.py

Language: Python

PDFTextExtractor

Purpose: This class is used for extracting text from PDF files.

Key Methods:
• extract_text(pdf_file) - Returns the extracted text from the given PDF file.
• save_text_to_file(text, output_file) - Saves the extracted text into a specified output file.
• extract_and_save(pdf_file, output_file) - Extracts text from a PDF file and saves it to an output file.

Quick Start:

```
# 1. Initialize
extractor = PDFTextExtractor()

# 2. Basic operations
text = extractor.extract_text("sample.pdf")
print(text)  # Output: 'This is a sample PDF text...'

# 3. Complete workflow
extractor.extract_and_save("sample.pdf", "output.txt")

# 4. Error handling
try:
    extractor.extract_text("nonexistent.pdf")
except FileNotFoundError:
    print("PDF file does not exist.")
```

Integration Examples:

```
# With file system
import os
pdf_files = [f for f in os.listdir() if f.endswith('.pdf')]
extractor = PDFTextExtractor()
for pdf_file in pdf_files:
    output_file = pdf_file.replace('.pdf', '.txt')
    extractor.extract_and_save(pdf_file, output_file)

# With web scraping
import requests
from bs4 import BeautifulSoup
url = "http://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
pdf_links = [a['href'] for a in soup.find_all('a', href=True) if a['href'].endswith('.pdf')]
extractor = PDFTextExtractor()
for pdf_link in pdf_links:
    response = requests.get(pdf_link)
    with open('temp.pdf', 'wb') as f:
        f.write(response.content)
    text = extractor.extract_text('temp.pdf')
    print(text)
```

Common Use Cases:
• **Data extraction:** Extracting text from PDF files for data analysis.
• **Web scraping:** Extracting text from PDF files found on web pages.
• **File conversion:** Converting PDF files to text files.

Performance Tips:
• Use multiprocessing or threading for extracting text from multiple PDF files concurrently.
• Delete temporary files after extraction to save disk space.

Troubleshooting:
• **"FileNotFoundError":** Check if the PDF file exists and the file path is correct.
• **"PermissionError":** Check if you have the necessary permissions to read the PDF file and write the output

file.
• **"PDFSyntaxError":** The PDF file may be corrupted or not a real PDF file.

Classes in pdf_crawler.py

Language: Python

PDFTextExtractor

Purpose: This class extracts text from PDF files, including text embedded within images.

Key Methods:
• __init__(openai_api_key: str) - Initializes the class with OpenAI API key.
• safe_gpt4o_mini_extract_text(image: Image.Image, retries: int = 5, sleep_seconds: int = 10) -> str - Safely extracts text from an image using OpenAI's GPT-4o-mini model, with retry logic for rate limits.
• gpt4o_mini_extract_text(image: Image.Image) -> str - Extracts text from an image using OpenAI's GPT-4o-mini model.
• extract_text_from_pdf(pdf_path: str) - Extracts text from all pages of a PDF file, including text within images.

Quick Start:

```
# 1. Initialize with OpenAI API key
extractor = PDFTextExtractor(openai_api_key="your_key")

# 2. Extract text from a PDF
pdf_text = extractor.extract_text_from_pdf("path_to_your_pdf")
print(pdf_text)  # Output: List of dictionaries with page number, extracted text, and OCR text from images

# 3. Extract text from an image within a PDF
from PIL import Image
image = Image.open("path_to_your_image")
text = extractor.safe_gpt4o_mini_extract_text(image)
print(text)  # Output: Extracted text from the image
```

Integration Examples:

```
# With other file processing tasks
file_processor = FileProcessor()
pdf_extractor = PDFTextExtractor(openai_api_key="your_key")
file_processor.add_task(pdf_extractor.extract_text_from_pdf)

# With a web scraper
scraper = WebScraper()
pdf_extractor = PDFTextExtractor(openai_api_key="your_key")
scraper.add_post_processing(pdf_extractor.extract_text_from_pdf)
```

Common Use Cases:
• **Document processing:** Extracting text from PDFs for further analysis or processing.
• **Web scraping:** Post-processing of scraped PDFs to extract text.
• **Data extraction:** Extracting text from images within PDFs.

Performance Tips:
• Use a higher retries parameter if you're frequently hitting rate limits.
• Increase sleep_seconds if you're still hitting rate limits after increasing retries.

Troubleshooting:
• **"RateLimitError":** Increase the retries and/or sleep_seconds parameters in safe_gpt4o_mini_extract_text().
• **"FileNotFoundError":** Check the path provided to extract_text_from_pdf().
• **"Permission denied":** Check your OpenAI API key and ensure it has the correct permissions.

Comprehensive Developer Guide

> **Complete Guide**: This section provides in-depth explanations with practical examples for modifying and extending the code.

__init__.py

Language: Python

What & Why

• **Purpose:** This __init__.py file is used to initialize the Python package and make the PDFTextExtractor class available at the package level.
• **Used for:**
- Importing the PDFTextExtractor class directly from the package.
- Organizing the package structure.
- Controlling what is exposed to the outside when the package is imported.
• **Part of:** This file is part of the root directory of the Python package.

Quick Start

```
# Basic usage pattern
from your_package import PDFTextExtractor

# Example 1: Most common use
extractor = PDFTextExtractor("path_to_your_pdf_file.pdf")
text = extractor.extract_text()
print(text)  # Output: "Extracted text from the PDF file"

# Example 2: Class usage
extractor = PDFTextExtractor()
extractor.load_file("path_to_your_pdf_file.pdf")
text = extractor.extract_text()
print(text)  # Output: "Extracted text from the PDF file"
```

Key Functions

• PDFTextExtractor(file_path) → returns an instance of PDFTextExtractor, use when you want to extract text from a PDF file.
• PDFTextExtractor.load_file(file_path) → loads a PDF file, use when you want to load a PDF file after creating an instance of PDFTextExtractor.
• PDFTextExtractor.extract_text() → extracts text from the loaded PDF file, call after loading a PDF file.

Common Patterns

```
# Pattern 1: Load file → Extract text
extractor = PDFTextExtractor("path_to_your_pdf_file.pdf")
text = extractor.extract_text()

# Pattern 2: Error handling
try:
    extractor = PDFTextExtractor("non_existent_file.pdf")
except FileNotFoundError:
    print("The specified file does not exist.")
```

Troubleshooting

• **Error "FileNotFoundError":** Check if the specified PDF file exists and the file path is correct.
• **Slow performance:** Usually caused by large PDF files, try to split the file into smaller chunks.
• **Memory issues:** Avoid loading large files all at once, use a buffer or stream the file instead.

---

pdf_crawler.py

Language: Python

What & Why

• **Purpose:** The pdf_crawler.py script is designed to extract text from PDF files, including text embedded within images.
• **Used for:**

- Extracting text from PDFs for data analysis or processing.
- Performing Optical Character Recognition (OCR) on images within PDFs.
- Automating the process of text extraction from large numbers of PDF files.
  • **Part of:** This script can be a component of any system that requires text extraction from PDFs, such as data mining, machine learning, or content management systems.

Quick Start

```
# Basic usage pattern
from pdf_crawler import PDFTextExtractor

# Create an instance of the PDFTextExtractor class with your OpenAI API key
extractor = PDFTextExtractor("your_openai_api_key")

# Extract text from a PDF file
result = extractor.extract_text_from_pdf("path_to_your_pdf_file")
print(result)  # Output: List of dictionaries containing page numbers, extracted text, and OCR text from images
```

Key Functions

• PDFTextExtractor(openai_api_key: str) → Initializes the PDFTextExtractor class with your OpenAI API key.
• safe_gpt4o_mini_extract_text(image: Image.Image, retries: int = 5, sleep_seconds: int = 10) -> str → Safely attempts to extract text from an image using the OpenAI GPT-4o-mini model, with optional parameters for retries and sleep time between retries.
• gpt4o_mini_extract_text(image: Image.Image) -> str → Extracts text from an image using the OpenAI GPT-4o-mini model.
• extract_text_from_pdf(pdf_path: str) → Extracts text from a PDF file, including text from images within the PDF.

Common Patterns

```
# Pattern 1: Extracting text from multiple PDFs
pdf_files = ["file1.pdf", "file2.pdf", "file3.pdf"]
extractor = PDFTextExtractor("your_openai_api_key")
for pdf in pdf_files:
    result = extractor.extract_text_from_pdf(pdf)
    print(result)

# Pattern 2: Error handling when extracting text from images
try:
    text = extractor.safe_gpt4o_mini_extract_text(image)
except RateLimitError:
    print("Rate limit hit. Please wait and try again.")
```

Troubleshooting

• **Error "RateLimitError":** This error occurs when the OpenAI API rate limit is hit. Increase the sleep_seconds parameter in safe_gpt4o_mini_extract_text() or reduce the frequency of your requests.
• **Slow performance:** This can be caused by large PDF files or a large number of images within the PDFs. Consider breaking your PDFs into smaller chunks or reducing the number of images.
• **Memory issues:** These can occur when processing large PDF files or images. Ensure your system has sufficient memory to handle the size of your PDFs and images.

Data Models & Database Documentation

Database Guide: Data structures, relationships, queries, and database interaction patterns.

Data models documentation will be generated when model files are detected.

Integration & Workflow Guides

Integration Patterns: External APIs, databases, queues, and operational workflows.

Integration guides will be generated when integration files are detected.

Troubleshooting & Common Issues

Quick Fixes: Common problems new developers encounter, with step-by-step solutions and prevention tips.

Common Issues

Issue	Solution
Build failures	Check dependencies and environment setup
Import errors	Verify all required packages are installed
Permission errors	Ensure proper file permissions

Getting Help

- Check the project README for additional setup instructions
- Review error logs for specific error messages
- Consult the project's issue tracker or documentation