perplexity

Look up with code examples the best practice for 2025 how to send a pdf to open ai model. I want to send a rubric and work, both as a pdf and get feedback based on the rubric, but I want technical implementation details

As of February 2025, processing PDFs with OpenAI models has become more streamlined with several effective approaches. This report provides comprehensive technical details for implementing PDF processing with OpenAI, focusing specifically on your use case of rubric-based assessment.

Overview of PDF Processing Methods

In 2025, there are four primary methods for processing PDFs with OpenAI models, each with distinct advantages depending on your specific requirements. The landscape has evolved significantly from earlier implementations, with the Assistants API now offering robust PDF handling capabilities.

OpenAI Assistants API with File Search Tool

The Assistants API provides the most direct and efficient method for PDF processing in 2025, particularly for documents containing both text and simple visuals. This approach is well-suited for your rubric assessment use case.

Technical Implementation

```
import openai
import os

# Initialize the OpenAI client
client = openai.OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

# Upload the PDF files
rubric_file = client.files.create(
    file=open("rubric.pdf", "rb"),
    purpose='assistants'
)

work_file = client.files.create(
    file=open("student_work.pdf", "rb"),
    purpose='assistants'
)
```

```
# Create an assistant specialized for rubric assessment
assistant = client.beta.assistants.create(
    name="Rubric Assessment Assistant",
    model="gpt-4o",
    instructions="You are an expert at evaluating work based on provided rubrics. When gi
    tools=[{"type": "file_search"}]
)

# Create a thread
thread = client.beta.threads.create()

# Add message with assessment instructions
await client.beta.threads.messages.create(
    thread.id,
    role="user",
    content="Please evaluate the student work against the provided rubric. Offer detailed
    file_ids=[rubric_file.id, work_file.id]
)

# Run the assistant
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id
)

# Check for completion and retrieve results
run_status = client.beta.threads.runs.retrieve(
    thread_id=thread.id,
    run_id=run.id
)

# Wait for completion (in production, implement proper polling)
# Then retrieve messages
messages = client.beta.threads.messages.list(
    thread_id=thread.id
)

# Process and display the assistant's response
for message in messages.data:
    if message.role == "assistant":
        print(message.content[^0].text.value)
```

This method allows direct processing of PDF files without manual text extraction, making it ideal for your rubric-based assessment needs[1] [2] [3].

Text Extraction and Direct API Approach

For cases where you need more control over the extraction process or when working with highly structured PDFs, extracting text programmatically before sending to the API remains viable in 2025.

Technical Implementation

```
import PyPDF2
import openai
import os

# Initialize the OpenAI client
client = openai.OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

def extract_text_from_pdf(pdf_path):
    """Extract text content from a PDF file."""
    text = ""
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        for page in reader.pages:
            text += page.extract_text()
    return text

# Extract text from both PDFs
rubric_text = extract_text_from_pdf("rubric.pdf")
work_text = extract_text_from_pdf("student_work.pdf")

# Combine with appropriate context
prompt = f"""
I need to evaluate student work based on a rubric.

RUBRIC:
{rubric_text}

STUDENT WORK:
{work_text}

Please evaluate the student work against each criterion in the rubric.
Provide specific feedback on strengths and areas for improvement for each criterion.
"""

# Send to chat completions API
response = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "You are an expert educator who specializes in prov
        {"role": "user", "content": prompt}
    ],
    temperature=0.2
)

print(response.choices[^0].message.content)
```

This approach gives you precise control over the text extraction process but may miss visual elements in the PDFs[4] [5].

Vector Database Approach for RAG

For complex, lengthy PDFs or when you need to implement a more sophisticated retrieval system, the Retrieval Augmented Generation (RAG) approach using vector databases has become a standard practice in 2025.

Technical Implementation

```
import PyPDF2
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chat_models import ChatOpenAI
from langchain.chains import ConversationalRetrievalChain
import os

# Set OpenAI API key
os.environ["OPENAI_API_KEY"] = "your-api-key"

def extract_text_from_pdf(pdf_path):
    """Extract text from PDF."""
    text = ""
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        for page in reader.pages:
            text += page.extract_text()
    return text

def create_vector_store(pdf_text):
    """Create a vector store from PDF text."""
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=500,  # Smaller chunks for embedding
        chunk_overlap=50  # Minimal overlap to reduce token count
    )

    chunks = text_splitter.split_text(pdf_text)
    embeddings = OpenAIEmbeddings()  # Uses your OPENAI_API_KEY
    vector_store = FAISS.from_texts(chunks, embeddings)
    return vector_store

def create_conversational_chain(vector_store):
    """Create a conversational chain for QA."""
    llm = ChatOpenAI(
        model="gpt-4o",
        temperature=0.2
    )

    retriever = vector_store.as_retriever(
        search_type="similarity",
        search_kwargs={"k": 5}
    )

    qa_chain = ConversationalRetrievalChain.from_llm(llm, retriever)
    return qa_chain
```

```
# Process the rubric and work
rubric_text = extract_text_from_pdf("rubric.pdf")
work_text = extract_text_from_pdf("student_work.pdf")

# Create vector stores
rubric_vectors = create_vector_store(rubric_text)
work_vectors = create_vector_store(work_text)

# Combine the vector stores (optional, depending on implementation)
# For simplicity, we'll use the rubric store for retrieval in this example
qa_chain = create_conversational_chain(rubric_vectors)

# Example conversation
chat_history = []
query = f"I have a student submission with the following content: {work_text[:1000]}... F

result = qa_chain({"question": query, "chat_history": chat_history})
print(result["answer"])
```

This method excels when dealing with lengthy documents where relevant information needs to be retrieved contextually[5] [6].

Vision Model Approach for Scanned Documents

For PDFs that are primarily images or scanned documents, converting to images and using OpenAI's vision capabilities has become a reliable approach in 2025.

Technical Implementation

```
import fitz  # PyMuPDF
import os
import io
from PIL import Image
import base64
import openai

# Initialize OpenAI client
client = openai.OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

def pdf_to_images(pdf_path):
    """Convert PDF pages to images."""
    images = []
    pdf_document = fitz.open(pdf_path)

    for page_number in range(pdf_document.page_count):
        page = pdf_document.load_page(page_number)
        pix = page.get_pixmap(matrix=fitz.Matrix(300/72, 300/72))
        img = Image.open(io.BytesIO(pix.tobytes()))

        # Convert to base64
        img_byte_arr = io.BytesIO()
        img.save(img_byte_arr, format='PNG')
        img_byte_arr = img_byte_arr.getvalue()
        encoded_img = base64.b64encode(img_byte_arr).decode('ascii')
```

```
        images.append(encoded_img)

    return images

# Convert both PDFs to images
rubric_images = pdf_to_images("rubric.pdf")
work_images = pdf_to_images("student_work.pdf")

# Prepare image content for the API
image_content = []

# Add rubric images
for idx, img in enumerate(rubric_images):
    image_content.append({
        "type": "image_url",
        "image_url": {
            "url": f"{img}",
            "detail": "high"
        }
    })
    if idx == 0:  # After first image, add a separator
        image_content.append({
            "type": "text",
            "text": "Above is the rubric. Below is the student work:"
        })

# Add work images
for img in work_images:
    image_content.append({
        "type": "image_url",
        "image_url": {
            "url": f"{img}",
            "detail": "high"
        }
    })

# Add final instruction text
image_content.append({
    "type": "text",
    "text": "Please evaluate the student work against each criterion in the rubric. Provi
})

# Send to the vision model
response = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {
            "role": "system",
            "content": "You are an expert educator specializing in assessment. Evaluate s
        },
        {
            "role": "user",
            "content": image_content
        }
    ],
```

```
    temperature=0.2
)

print(response.choices[^0].message.content)
```

This approach is particularly effective for scanned documents or PDFs with significant visual elements that text extraction might miss[2] [3].

Implementing Rubric-Based Assessment

For your specific use case of rubric-based assessment, the Assistants API approach (Method 1) is recommended as the most straightforward implementation in 2025. However, here's a tailored implementation that combines elements of multiple approaches for robustness:

```
import openai
import os
import PyPDF2
import json

# Initialize OpenAI client
client = openai.OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

# Function to extract text from PDF
def extract_text_from_pdf(pdf_path):
    text = ""
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        for page in reader.pages:
            text += page.extract_text()
    return text

# 1. First approach: Upload PDFs to Assistant
rubric_file = client.files.create(
    file=open("rubric.pdf", "rb"),
    purpose='assistants'
)

work_file = client.files.create(
    file=open("student_work.pdf", "rb"),
    purpose='assistants'
)

# Create a specialized assistant for rubric assessment
assistant = client.beta.assistants.create(
    name="Rubric Assessment Assistant",
    model="gpt-4o",
    instructions="""
    You are an expert at evaluating work based on provided rubrics.
    When given both a rubric and student work, analyze the work against each criterion in
    For each criterion in the rubric:
    1. Provide a score based on the rubric's scoring system
    2. Explain the reasoning for the score with specific examples from the student's work
    3. Offer constructive feedback for improvement
    4. Highlight strengths demonstrated in the student's work
```

```
    Format your response as structured JSON with the following schema:
    {
        "overall_assessment": "Summary of the overall evaluation",
        "total_score": "Total score if applicable",
        "criteria_evaluations": [
            {
                "criterion_name": "Name of criterion from rubric",
                "score": "Score for this criterion",
                "justification": "Explanation of the score with specific examples",
                "strengths": "Identified strengths",
                "areas_for_improvement": "Constructive feedback"
            }
        ]
    }
    """,
    tools=[{"type": "file_search"}]
)

# Create thread
thread = client.beta.threads.create()

# Add message with assessment instructions
client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content="Please evaluate the student work against the rubric, following the structure
    file_ids=[rubric_file.id, work_file.id]
)

# Run the assistant
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id
)

# In production code, implement proper polling for completion
# For demonstration, assuming run is complete:

# Retrieve and parse results
messages = client.beta.threads.messages.list(
    thread_id=thread.id
)

assessment_result = None
for message in messages.data:
    if message.role == "assistant":
        try:
            # Extract and parse JSON from response
            content_text = message.content[^0].text.value
            # Find JSON in content (assuming it's properly formatted)
            start_idx = content_text.find('{')
            end_idx = content_text.rfind('}') + 1
            if start_idx >= 0 and end_idx > start_idx:
                json_content = content_text[start_idx:end_idx]
                assessment_result = json.loads(json_content)
```

```
                break
        except Exception as e:
            print(f"Error parsing assessment result: {e}")

# Process the assessment result
if assessment_result:
    print(f"Overall Assessment: {assessment_result['overall_assessment']}")
    print(f"Total Score: {assessment_result['total_score']}")
    print("\nDetailed Criteria Evaluations:")
    for eval in assessment_result['criteria_evaluations']:
        print(f"\n{eval['criterion_name']} - Score: {eval['score']}")
        print(f"Justification: {eval['justification']}")
        print(f"Strengths: {eval['strengths']}")
        print(f"Areas for Improvement: {eval['areas_for_improvement']}")
```

This implementation leverages the Assistants API with structured output requirements and error handling for a robust assessment system[1] [2] [3].

Considerations and Limitations

When implementing PDF processing with OpenAI models in 2025, several important considerations remain:

1. **PDF Complexity**: Complex PDFs with intricate layouts, mathematical notations, or tables may still require specialized pre-processing. For such cases, combining methods (e.g., using both text extraction and vision approaches) might yield better results[2] [3].
2. **Scanned Documents**: For scanned PDFs without searchable text, OCR pre-processing or the vision approach is necessary. As of 2025, OCR before sending to OpenAI's models remains a recommended practice for scanned documents[2].
3. **Security Concerns**: For secure PDFs with access restrictions, specialized handling may be required before processing with OpenAI. Consider security implications when extracting text from secured documents[2].
4. **Token Limitations**: PDFs containing extensive text may exceed token limits. In such cases, the RAG approach with vector stores is recommended for managing large documents efficiently[5].
5. **Model Selection**: As of February 2025, GPT-4o offers comprehensive capabilities for PDF processing, including both text and visual elements. For budget-conscious implementations, GPT-4o-mini provides a balance of performance and cost efficiency[2].

Conclusion

The landscape of PDF processing with OpenAI models has evolved significantly by 2025. The Assistants API with file_search capability offers the most straightforward approach for most use cases, including rubric-based assessments. For specialized needs or complex documents, combining approaches (text extraction, vector databases, or vision models) provides robust solutions.

For your specific rubric-based assessment use case, the implementation provided leverages the Assistants API for direct PDF processing while structuring the output to facilitate automated

assessment workflows. This approach balances technical simplicity with robust functionality, making it well-suited for educational assessment applications in 2025.

⁂

1. https://community.openai.com/t/best-practices-for-pdf-parsing-with-assistants-api-and-file-search-tool/1105328
2. https://community.openai.com/t/best-practice-scanned-pdf-what-model-to-use/1122717
3. https://stackoverflow.com/questions/77469097/how-can-i-process-a-pdf-using-openais-apis-gpts
4. https://community.openai.com/t/chat-completions-api-for-attach-a-pdf/782411
5. https://community.openai.com/t/best-way-to-interact-with-pdf-2025/1098423
6. https://www.youtube.com/watch?v=s6k5Ol7BQGg