

# Guía de uso de Mistral OCR

- [1. Descripción](#)
- [2. Objetivo](#)
- [3. Fuentes de datos aceptadas](#)
- [4. Tipos de salidas](#)
- [5. Funcionalidades y casos de uso](#)
- [6. Guía de uso por funcionalidad](#)
  - [6.0 Definiciones](#)
  - [6.1 Setup inicial](#)
  - [6.2 Inicialización del cliente](#)
  - [6.3 Mistral OCR](#)
    - [6.3.1 OCR básico \(Without Batch\)](#)
    - [6.3.2 OCR Masivo \(With Batch Inference\)](#)
  - [6.4 Anotaciones](#)
    - [6.4.1 Mistral OCR sin Anotaciones](#)
    - [6.4.2 Mistral OCR con Anotaciones](#)
    - [6.4.3 Documento completo con Anotaciones](#)
  - [6.5 Document QnA Cookbook](#)
    - Ejemplos de Prompts para el caso práctico.
  - [6.6 OCR Cookbook](#)
- [7. Otros tópicos \(basado en experiencia\)](#)
  - [7.1 Costos y rendimiento](#)
  - [7.2 Despliegue y escalabilidad](#)
  - [7.3 Preprocesamiento de imágenes](#)
  - [7.4 Seguridad y cumplimiento](#)
  - [7.5 Monitorización y alertas](#)

## 1. Descripción

Mistral OCR es una solución de OCR multimodal basada en la nube que combina alta precisión en la extracción de texto con capacidades avanzadas de segmentación de tablas, ecuaciones y gráficos. Permite procesar tanto imágenes (JPEG, PNG, TIFF) como documentos PDF, devolviendo salidas estructuradas para integrarse fácilmente en pipelines de IA, RAG o sistemas de análisis de documentos.

## 2. Objetivo

### 3. Fuentes de datos aceptadas

- **Formatos:** PDFs y principales formatos de imagen (JPEG, PNG, TIFF) como inputs directos.
- **Vías de envío:**

- Mediante URL ( `document_url` , `image_url` )
- Cadena Base64 incluida en el cuerpo JSON.
- **Límites:** Archivos hasta 50 MB y 1000 páginas. Para volúmenes mayores, emplear **Batch Inference**

#### 4. Tipos de salidas

- **Texto intercalado:** JSON con bloques de texto ordenados, con saltos de línea y posición en la página.
- **Imágenes embebidas:** Base64 para segmentos gráficos enlazados a su bounding box.
- **Metadatos estructurados:** bounding boxes, clasificación de contenido (texto, tabla, ecuación, gráfico) y número de página.
- **Markdown automático:** conversión de páginas a Markdown lista para ingestión en LLMs.

#### 5. Funcionalidades y casos de uso

- **OCR básico:** extracción de texto y gráficos con precisión líder (~99 %) en múltiples idiomas
- **Batch Inference:** procesamiento paralelo de cientos a miles de documentos con 50 % ahorro de coste.
- **Detección de tablas y ecuaciones:** segmenta estructuras complejas para análisis técnico o científico.
- **Annotations:** añadir metadatos semánticos personalizados en JSON según tu esquema.
- **RAG / Document QnA:** integración con pipelines de Retrieval-Augmented Generation y QnA basados en LLM.

#### 6. Guía de uso por funcionalidad

##### 6.0 Definiciones

###### 1. Formato Markdown

Markdown es un **lenguaje de marcado ligero** que usa **texto plano** con una sintaxis mínima para indicar **títulos** (con `#`), **negritas** (`**texto**`), **cursivas** (`_texto_`), **listas**, **enlaces**, **imágenes**, etc. por ejemplo:

```
# Título de la página
```

```
Este es un párrafo con **negrita**, _cursiva_ y una imagen:
```

```
![img-0.jpeg](data:image/jpeg;base64,ABC123...)
```

Formato Markdown

Al exportar ese campo, se guarda no solo el texto plano, sino también su estructura (títulos, saltos de línea, imágenes incrustadas) lista para verse bien en visores de Markdown o en sistemas que soporten ese formato.

Entonces a lo largo del documento tendré dos frentes con respecto a este formato:

- **Como objeto JSON** (`response.pages[0].markdown`): es una **cadena** que contiene esa sintaxis (por ejemplo `"# Título\n\nEste es **negrita**\n\n![img-0.jpeg](data:image/jpeg;base64,ABC123...)"`).
- **Como renderizado** (`display(Markdown(...))`): esa misma cadena se interpreta y muestra con estilos (encabezados grandes, texto en negrita, las imágenes incrustadas) en cualquier visor compatible con Markdown.

En Mistral OCR, el campo `markdown` no es “HTML” ni “texto plano sin formato”, sino esa **sintaxis Markdown** que puede guardarse tal cual en JSON y, cuando se desee, procesarse para mostrarse bonito en una UI o convertirse a HTML.

## 2. Codificación Base64

La codificación Base64 es un método que convierte **datos binarios** (como el contenido de una imagen o un PDF) en una **cadena de texto ASCII** segura para su transmisión en JSON, XML o URLs.

- **Paso 1: Lectura de bytes**

El archivo (imagen o PDF) se carga en memoria y se lee como una secuencia de bytes.

- **Paso 2: Codificación a Base64**

Esos bytes se transforman en un flujo de caracteres del alfabeto Base64 (64 símbolos imprimibles: A–Z, a–z, 0–9, “+” y “/”) generando otra secuencia de bytes, agregando “=” como relleno cuando es necesario.

- **Paso 3: Decodificación a UTF-8**

La secuencia de bytes resultante de la codificación se decodifica en **UTF-8**, obteniendo una **cadena de texto** que puede incrustarse directamente en JSON (`"data:image/jpeg;base64,..."`) o URLs seguras.

### 6.1 Setup inicial

Instalar librerías y preparar el entorno de ejecución.

```
1 pip install mistralai datasets
```

Este comando instala el cliente de Mistral y la librería `datasets` para cargar ejemplos.

## 6.2 Inicialización del cliente

Crear el objeto `client` con tu API key y definir el modelo OCR.

```
1 from mistralai import Mistral
2 client = Mistral(api_key="TU_API_KEY")
3 ocr_model = "mistral-ocr-latest"
```

El cliente gestiona las llamadas HTTP y `ocr_model` indica la variante de OCR a usar.

## 6.3 Mistral OCR

### 6.3.1 OCR básico (Without Batch)

OCR básico (Without Batch) Procesamiento uno a uno para entornos de desarrollo o volúmenes pequeños.

#### Codificación Base64

Convertir imagen o PDF en cadena de texto para enviarla en JSON.

```
1 import base64
2 from io import BytesIO
3 from PIL import Image
4
5 def encode_image_data(image_data):
6     try:
7         # Asegurar que cada image_data sea bytes
8         if isinstance(image_data, bytes):
9             # Codificar bytes directamente en base64
10            return base64.b64encode(image_data).decode('utf-8')
11        else:
12            # Convertir los datos de la imagen a bytes si aún no lo están
13            buffered = BytesIO()
14            image_data.save(buffered, format="JPEG")
15            return base64.b64encode(buffered.getvalue()).decode('utf-8')
16    except Exception as e:
17        print(f"Error encoding image: {e}")
18        return None
```

Paso 1: Leer bytes; Paso 2: Codificar a Base64; Paso 3: Decodificar a UTF-8.

#### Instanciar los datos

Cargar un subconjunto de 100 imágenes de ejemplo del dataset HuggingFaceM4/DocumentVQA:

```
1 from datasets import load_dataset
2
```

```
3 n_samples = 100
4 dataset = load_dataset("HuggingFaceM4/DocumentVQA", split="train", streaming=True)
5 subset = list(dataset.take(n_samples))
```

Se usa streaming para no descargar todo el dataset.

## Bucle de inferencia

Con el subconjunto de 100 muestras, se recorre cada imagen para extraer el texto y se guarda los resultados en un nuevo conjunto de datos en formato markdown.

```
1 from tqdm import tqdm
2
3 ocr_dataset = []
4 for sample in tqdm(subset):
5     image_data = sample['image'] # 'image' contains the actual image data
6
7     # Codificar los datos de la imagen en base64
8     base64_image = encode_image_data(image_data)
9     image_url = f"data:image/jpeg;base64,{base64_image}"
10
11    # Procesar la imagen usando Mistral OCR
12    response = client.ocr.process(
13        model=ocr_model,
14        document={
15            "type": "image_url",
16            "image_url": image_url,
17        }
18    )
19
20    # Almacene los datos de la imagen y el contenido de OCR en el nuevo conjunto de datos
21    ocr_dataset.append({
22        'image': base64_image,
23        'ocr_content': response.pages[0].markdown # Since we are dealing with single images, there
24    })
```

Cada iteración llama al endpoint y almacena la salida en formato markdown.

En Mistral OCR, la propiedad `response.pages[0].markdown` devuelve el texto extraído en formato Markdown, para indicar para indicar **títulos** (con `#`), **negritas** (`**texto**`), **cursivas** (`_texto_`), **listas**, **enlaces**, **imágenes**, etc.

## Exportar resultados

Los resultados se exportan como un archivo JSONL (una línea JSON por registro).

```
1 import json
2
3 with open('ocr_dataset.jsonl', 'w') as f:
4     for record in ocr_dataset:
5         f.write(json.dumps(record) + "\n")
```

El archivo `ocr_dataset.jsonl` contendrá una línea JSON por imagen, con su Markdown

Con JSONL se procesa cada registro sin cargar todo en memoria, y con JSON “array” se tendrá un fichero más legible si se abre en un editor.

Ejemplo de tres líneas en `ocr_results.jsonl`:

```
1 {"image": "...base64...", "ocr_content": "# Página 1\nTexto extraído..."}  
2 {"image": "...base64...", "ocr_content": "# Página 2\nMás texto..."}  
3 {"image": "...base64...", "ocr_content": "# Página 3\nOtra página..."}
```

El valor de la clave `"ocr_content"` es precisamente la cadena en **Markdown** que generó el OCR (títulos, saltos de línea, imágenes embebidas, etc.).

#### 6.3.2 OCR Masivo (With Batch Inference)

OCR masivo (With Batch Inference) Procesamiento en lotes a gran escala con ahorro de coste.

Para usar la inferencia por lotes, se necesita crear un archivo JSONL que contenga los datos de cada una de las imágenes.

#### Definir la función para crear el archivo JSONL

Se define la función `create_batch_file` para crear un archivo JSONL donde cada línea representa una solicitud independiente. Este archivo servirá como entrada al servicio de Batch Inference.

```
1 def create_batch_file(image_urls, output_file):  
2     with open(output_file, 'w') as file:  
3         for index, url in enumerate(image_urls):  
4             entry = {  
5                 "custom_id": str(index),  
6                 "body": {  
7                     "document": {  
8                         "type": "image_url",  
9                         "image_url": url  
10                    },  
11                    "include_image_base64": True  
12                }  
13            }  
14            file.write(json.dumps(entry) + '\n')
```

La función genera un archivo `output_file` listo para subir como batch.

#### Codificar las imágenes

Este fragmento de código recorre un conjunto de datos de imágenes, codifica cada imagen en Base64 y genera una URL de datos para cada una. Las URLs resultantes se almacenan en la lista `image_urls`, que se puede utilizar posteriormente para crear un archivo JSONL o para otros propósitos en la aplicación.

```
1 image_urls = []
2 for sample in tqdm(subset):
3     image_data_bytes = sample['image']
4     base64_image = encode_image_data(image_data_bytes)
5     image_url = f"data:image/jpeg;base64,{base64_image}"
6     image_urls.append(image_url)
```

El array `image_urls` contiene todas las entradas necesarias para crear el JSONL.

### Proceso de carga de archivo para procesamiento por lotes

En esta sección, se define la variable `batch_file` con el nombre del archivo `batch_file.jsonl`, que es un archivo en formato JSON Lines (JSONL) destinado a contener solicitudes para el procesamiento por lotes. Luego, se llama a la función `create_batch_file`, pasando `image_urls` (la lista de URLs de imágenes codificadas en Base64) y `batch_file` para crear y poblar el archivo JSONL. Finalmente, el archivo se carga en la API de Mistral OCR.

La respuesta de la API se almacena en la variable `batch_data`, que contiene información sobre la carga, incluyendo un identificador único para futuras referencias.

```
1 batch_file = "batch_file.jsonl"
2 create_batch_file(image_urls, batch_file)
3
4 batch_data = client.files.upload(
5     file={
6         "file_name": batch_file,
7         "content": open(batch_file, "rb")},
8     purpose = "batch"
9 )
```

Se obtiene `batch_data.id` para referenciar este archivo en la creación del job.

### Crear un job e iniciar el proceso

Para iniciar el proceso, se crea el job de Batch Inference y se supervisa su estado hasta su finalización.

```
1 created_job = client.batch.jobs.create(
2     input_files=[batch_data.id],
3     model=ocr_model,
4     endpoint="/v1/ocr",
5     metadata={"job_type": "testing"}
6 )
7
8 # Polling automático del estado del job
9 while True:
10     status = client.batch.jobs.get(job_id=created_job.id).status
11     print(f"Estado del job: {status}")
12     if status == "SUCCESS":
13         break
14     time.sleep(2)
```

El bucle se repite hasta que el job finaliza con éxito ( `SUCCESS` ).

## Descarga de resultados

Al completarse el procesamiento, se descarga el archivo de salida que contiene todas las respuestas.

```
1 downloaded = client.files.download(file_id=created_job.output_file)
2 # `downloaded` es un objeto que incluye el JSONL con cada respuesta de OCR
```

El fichero descargado presenta una línea JSON por imagen, incluyendo el markdown y sus metadatos correspondientes.

## Optimización de costos y rendimiento

Ahorro de coste del 50 % frente al OCR individual y throughput de miles de páginas por dólar

Todo el flujo en el notebook “Apply OCR to Convert Images into Text” del Cookbook”:

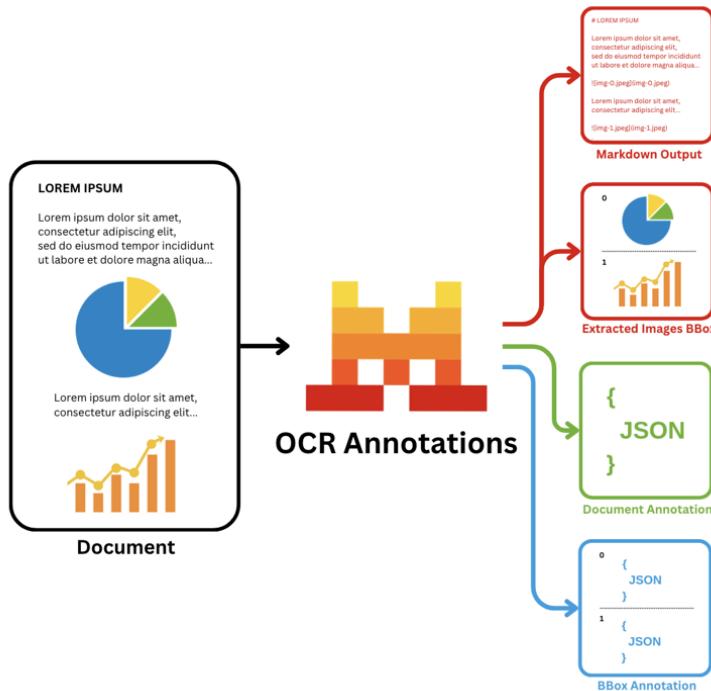
[cookbook/mistral/ocr/batch\\_ocr.ipynb at main · mistralai/cookbook](#)

## 6.4 Anotaciones

La funcionalidad de Anotaciones permite enriquecer la salida OCR con metadatos estructurados tanto del documento completo como de cada caja delimitadora (bbox).

La API de IA de Mistral Document añade dos funciones de anotación:

- **document\_annotation:** Devuelve la anotación de todo el documento según el esquema de entrada
- **bbox\_annotation:** Proporciona anotaciones específicas para cada región detectada (tablas, gráficos, texto), permitiendo describir o categorizar cada figura. Por ejemplo, el usuario puede solicitar que se describa o subtitule una figura.



Formatos de salida de “OCR Annotations”

Para llevar a cabo esta funcionalidad se hace uso de los siguientes datos:

### Descargar PDF de ejemplo

```
1 wget https://raw.githubusercontent.com/mistralai/cookbook/refs/heads/main/mistral/ocr/mistral7b.pdf
```

#### 6.4.1 Mistral OCR sin Anotaciones

### Definir la Función para Codificar el PDF

Se define una función llamada `encode_pdf` que toma la ruta de un archivo PDF y lo codifica en Base64.

```
1 def encode_pdf(pdf_path):
2     """Encode the pdf to base64."""
3     try:
4         with open(pdf_path, "rb") as pdf_file:
5             return base64.b64encode(pdf_file.read()).decode('utf-8')
6     except FileNotFoundError:
7         print(f"Error: The file {pdf_path} was not found.")
8         return None
9     except Exception as e: # Added general exception handling
10        print(f"Error: {e}")
11        return None
```

### Especificar la Ruta del PDF

Se define la ruta del archivo PDF que se va a procesar.

```
1 pdf_path = "mistral7b.pdf"
```

### Codificar el PDF en Base64

Se llama a la función `encode_pdf` para obtener la representación en Base64 del PDF.

```
1 base64_pdf = encode_pdf(pdf_path)
```

## Llamar a la API de OCR

Se realiza una solicitud a la API de OCR de Mistral, pasando el PDF codificado en Base64 como un documento.

```
1 pdf_response = client.ocr.process(
2     model="mistral-ocr-latest",
3     document={
4         "type": "document_url",
5         "document_url": f"data:application/pdf;base64,{base64_pdf}"
6     },
7     include_image_base64=True
8 )
```

## Convertir la Respuesta a Formato JSON

La respuesta de la API se convierte a un diccionario JSON para facilitar su manipulación. Así mismo, se imprime una parte de la respuesta para verificar el contenido.

```
1 response_dict = json.loads(pdf_response.model_dump_json())
2 print(json.dumps(response_dict, indent=4)[0:1000]) # check the first 1000 characters
```

```
{
    "pages": [
        {
            "index": 0,
            "markdown": "# Mistral 7B \n\nAlbert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lampe, Lucile Saulnier, L\u00e9o Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth\u00e9e Lacroix, William El Sayed\n\n\n\n#### Abstract\n\nWe introduce Mistral 7B, a 7-billion-parameter language model engineered for superior performance and efficiency. Mistral 7B outperforms the best open 13B model (Llama 2) across all evaluated benchmarks, and the best released 34B model (Llama 1) in reasoning, math, and code generation. Our model leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length with a reduced inference cost. We also provide a model fine-tuned
    
```

1000 Caracteres de la Respuesta

## Definir Funciones para Manejar la Respuesta de OCR

Se definen dos funciones para reemplazar imágenes en el texto Markdown y combinar el texto y las imágenes en un solo documento Markdown.

- **Reemplazar Imágenes en Markdown:**

```
1 def replace_images_in_markdown(markdown_str: str, images_dict: dict) -> str:
2     """
3         Replace image placeholders in markdown with base64-encoded images.
4
5     Args:
6         markdown_str: Markdown text containing image placeholders
7         images_dict: Dictionary mapping image IDs to base64 strings
8
9     Returns:
10        Markdown text with images replaced by base64 data
11    """
12    for img_name, base64_str in images_dict.items():
13        markdown_str = markdown_str.replace(
```

```
14         f"![{img_name}](.{img_name})", f"![{img_name}](.{base64_str})"
15     )
16 return markdown_str
```

- **Combinar Markdown y Imágenes:**

```
1 def get_combined_markdown(ocr_response: OCRResponse) -> str:
2 """
3     Combine OCR text and images into a single markdown document.
4
5     Args:
6         ocr_response: Response from OCR processing containing text and images
7
8     Returns:
9         Combined markdown string with embedded images
10    """
11    markdowns: list[str] = []
12    # Extract images from page
13    for page in ocr_response.pages:
14        image_data = {}
15        for img in page.images:
16            image_data[img.id] = img.image_base64
17        # Replace image placeholders with actual images
18        markdowns.append(replace_images_in_markdown(page.markdown, image_data))
19
20    return "\n\n".join(markdowns)
```

### Mostrar el documento combinado renderizado en Markdown

Finalmente, Se utiliza la función `get_combined_markdown` para fusionar el texto extraído y las imágenes (en Base64), generando un único documento en sintaxis Markdown, que a continuación se renderiza con `display(Markdown(...))` para visualizar de forma estructurada y legible tanto los encabezados, párrafos y listas, como las figuras embebidas.

```
1 display(Markdown(get_combined_markdown(pdf_response)))
```

# Mistral 7B

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed



## Abstract

We introduce Mistral 7B, a 7-billion-parameter language model engineered for superior performance and efficiency. Mistral 7B outperforms the best open 13B model (Llama 2) across all evaluated benchmarks, and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. Our model leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length with a reduced inference cost. We also provide a model fine-tuned to follow instructions, Mistral 7B - Instruct, that surpasses Llama 2 13B - chat model both on human and automated

Visualización formateada (renderizado) del texto e imágenes extraídos mediante Markdown.

### 6.4.2 Mistral OCR con Anotaciones

#### Importar Módulos Necesarios

Se importan las clases necesarias de `pydantic` y `enum` que permitirán definir los esquemas de datos para las anotaciones.

```
1 from pydantic import BaseModel, Field
2 from enum import Enum
```

#### Definir el Enum para Tipos de Imagen

Se define una clase llamada `ImageType` que especifica los tipos de imágenes que se pueden encontrar en el documento.

```
1 class ImageType(str, Enum):
2     GRAPH = "graph"
3     TEXT = "text"
4     TABLE = "table"
5     IMAGE = "image"
```

#### Definir el Modelo de Imagen

Se crea una clase `Image`, que hereda de `BaseModel`. Cada instancia representará la anotación de una caja delimitadora (bbox), incluyendo su **tipo** y una **descripción**.

```
1 class Image(BaseModel):
2     image_type: ImageType = Field(..., description="The type of the image. Must be one of 'graph',")
3     description: str = Field(..., description="A description of the image.")
```

#### Definir el Modelo de Documento

Se crea una clase `Document` que también hereda de `BaseModel`, para anotar información global del archivo completo: **idioma detectado, lista de autores y resumen automático**.

```
1 class Document(BaseModel):
2     language: str = Field(..., description="The language of the document in ISO 639-1 code format")
3     summary: str = Field(..., description="A summary of the document.")
4     authors: list[str] = Field(..., description="A list of authors who contributed to the document")
```

## Preparar la Llamada a la API de OCR

Se importa la función necesaria que convierte los modelos Pydantic en el formato requerido por la API.

```
1 from mistralai.extra import response_format_from_pydantic_model
```

## Realizar la Llamada a la API de OCR con Anotaciones

Se invoca `client.ocr.process`, indicando:

- El **modelo OCR** a usar.
- Las **páginas** a procesar (límite de 8 para anotación global).
- El **documento** en Base64.
- El formato de anotación para **cajas delimitadoras** (bbox) y para el **documento**.
- `include_image_base64=False` para omitir las imágenes en la respuesta, y centrarse solo en los metadatos.

```
1 # OCR Call with Annotations
2 annotations_response = client.ocr.process(
3     model="mistral-ocr-latest",
4     pages=list(range(8)), # Document Annotations has a limit of 8 pages, we recommend splitting your document into multiple requests
5     document={
6         "type": "document_url",
7         "document_url": f"data:application/pdf;base64,{base64_pdf}"
8     },
9     bbox_annotation_format=response_format_from_pydantic_model(Image),
10    document_annotation_format=response_format_from_pydantic_model(Document),
11    include_image_base64=False # We are not interested on retrieving the bbox images in this example
12 )
```

- En la raíz de `annotations_response` aparece un único campo `document_annotation`, con el JSON (serializado) que sigue el esquema `Document` (campos `language`, `summary`, `authors`).
- Dentro de cada página (`response.pages[i]`), en el array `images`, cada objeto incluye un campo `image_annotation`, con el JSON (serializado) según el esquema `Image` (`image_type` y `description`).

Así, `document_annotation` se devuelve **una vez**, mientras que `image_annotation` aparece **por cada imagen** detectada.

## Parsear la cadena JSON a un dict de Python

Se convierte la respuesta serializada del OCR (una cadena JSON sin formato legible) en un objeto nativo de Python para poder acceder a sus campos directamente.

```
1 response_dict = json.loads(annotations_response.model_dump_json())
```

## Serializar el dict de Python a un JSON formateado para visualizarlo

Se toma el diccionario de Python y se genera una cadena JSON con sangrías (`indent=4`), de modo que resulte fácil de leer en consola o en logs.

```
1 print(json.dumps(response_dict, indent=4))
```

Estructura cruda de la respuesta OCR donde el texto extraído aparece bajo la clave `markdown`.

```
{
    "pages": [
        {
            "index": 0,
            "markdown": "# Mistral 7B\n\nAlbert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lampe, Lucile Saulnier, L\u00e9o Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth\u00e9e Lacroix, William El Sayed\n\n![img-0.jpeg](img-0.jpeg)\n\nAbstract\n\nWe introduce Mistral 7B, a 7-billion-parameter language model engineered for superior performance and efficiency. Mistral 7B outperforms the best open 13B model (Llama 2) across all evaluated benchmarks, and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. Our model leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length with a reduced inference cost. We also provide a model fine-tuned to follow instructions, Mistral 7B - Instruct, that surpasses Llama 2 13B - chat model both on human and automated benchmarks. Our models are released under the Apache 2.0 license. Code: https://github.com/mistralai/mistral-src Webpage: https://mistral.ai/news/announcing-mistral-7b/\n\nIn the rapidly evolving domain of Natural Language Processing (NLP), the race towards higher model performance often necessitates an escalation in model size. However, this scaling tends to increase computational costs and inference latency, thereby raising barriers to deployment in practical, real-world scenarios. In this context, the search for balanced models delivering both high-level performance and efficiency becomes critically essential. Our model, Mistral 7B, demonstrates that a carefully designed language model can deliver high performance while maintaining an efficient inference. Mistral 7B outperforms the previous best 13B model (Llama 2, [26]) across all tested benchmarks, and surpasses the best 34B model (LLaMA 34B, [25]) in mathematics and code generation. Furthermore, Mistral 7B approaches the coding performance of Code-Llama 7B [20], without sacrificing performance on non-code related benchmarks.\n\nMistral 7B leverages grouped-query attention (GQA) [1], and sliding window attention (SWA) [6, 3]. GQA significantly accelerates the inference speed, and also reduces the memory requirement during decoding, allowing for higher batch sizes hence higher throughput, a crucial factor for real-time applications. In addition, SWA is designed to handle longer sequences more effectively at a reduced computational cost, thereby alleviating a common limitation in LLMs. These attention mechanisms collectively contribute to the enhanced performance and efficiency of Mistral 7B.",
            "images": [
                {
                    "id": "img-0.jpeg",
                    "top_left_x": 425,
                    "top_left_y": 598,
                    "bottom_right_x": 1283,
                    "bottom_right_y": 893,
                    "image_base64": null,
                    "image_annotation": "{\n    \"image_type\": \"image\",\n    \"description\": \"A 3D rendering of the text 'Mistral AI' in a gradient of warm colors, transitioning from orange to brown.\n\"}\n"
                }
            ],
            "dimensions": {
                "dpp": 200,
                "height": 2200,
                "width": 1700
            }
        },
        {
            "index": 1,
            "markdown": "# Mistral 7B\n\nAlbert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lampe, Lucile Saulnier, L\u00e9o Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth\u00e9e Lacroix, William El Sayed\n\n![img-0.jpeg](img-0.jpeg)\n\nAbstract\n\nWe introduce Mistral 7B, a 7-billion-parameter language model engineered for superior performance and efficiency. Mistral 7B outperforms the best open 13B model (Llama 2) across all evaluated benchmarks, and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. Our model leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length with a reduced inference cost. We also provide a model fine-tuned to follow instructions, Mistral 7B - Instruct, that surpasses Llama 2 13B - chat model both on human and automated benchmarks. Our models are released under the Apache 2.0 license. Code: https://github.com/mistralai/mistral-src Webpage: https://mistral.ai/news/announcing-mistral-7b/\n\nIn the rapidly evolving domain of Natural Language Processing (NLP), the race towards higher model performance often necessitates an escalation in model size. However, this scaling tends to increase computational costs and inference latency, thereby raising barriers to deployment in practical, real-world scenarios. In this context, the search for balanced models delivering both high-level performance and efficiency becomes critically essential. Our model, Mistral 7B, demonstrates that a carefully designed language model can deliver high performance while maintaining an efficient inference. Mistral 7B outperforms the previous best 13B model (Llama 2, [26]) across all tested benchmarks, and surpasses the best 34B model (LLaMA 34B, [25]) in mathematics and code generation. Furthermore, Mistral 7B approaches the coding performance of Code-Llama 7B [20], without sacrificing performance on non-code related benchmarks.\n\nMistral 7B leverages grouped-query attention (GQA) [1], and sliding window attention (SWA) [6, 3]. GQA significantly accelerates the inference speed, and also reduces the memory requirement during decoding, allowing for higher batch sizes hence higher throughput, a crucial factor for real-time applications. In addition, SWA is designed to handle longer sequences more effectively at a reduced computational cost, thereby alleviating a common limitation in LLMs. These attention mechanisms collectively contribute to the enhanced performance and efficiency of Mistral 7B."
            }
        }
    ]
}
```

Respuesta JSON con campo “markdown” para **Image**

```

1,
"model": "mistral-ocr-2503-completion",
"usage_info": {
    "pages_processed": 8,
    "doc_size_bytes": 374978
},
"document_annotation": "{\n    \"language\": \"en\",\n    \"summary\": \"The document introduces Mistral 7B, a 7-billion-parameter language model designed for superior performance and efficiency. It outperforms the best open 13B model (Llama 2) across all evaluated benchmarks and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. The model uses grouped-query attention (GQA) for faster inference and sliding window attention (SWA) to handle sequences of arbitrary length with reduced inference cost. Mistral 7B is released under the Apache 2.0 license and includes a fine-tuned version, Mistral 7B - Instruct, which surpasses Llama 2 13B - Chat model on human and automated benchmarks. The document also discusses the architectural details, performance comparisons, and the efficiency of Mistral 7B.\",\n    \"authors\": [\n        \"Albert Q. Jiang\", \"Alexandre Sablayrolles\", \"Arthur Mensch\", \"Chris Bamford\", \"Devendra Singh Chaplot\", \"Diego de las Casas\", \"Florian Bressand\", \"Gianna Lengyel\", \"Guillaume Lamble\", \"Lucile Saunier\", \"Lelio Remond Lavaud\", \"Marie-Anne Lachaux\", \"Pierre Stock\", \"Tevon Le Scao\", \"Thibaut Lavril\", \"Thomas Wang\", \"Timoth\u00e9e Lacroix\", \"William El Sayed\"\n    ]\n}"
}

```

Respuesta JSON con campo “markdown” para **Document**

### Explicación de la estructura de la respuesta con anotaciones

A continuación se muestra un fragmento de la salida JSON tras llamar al endpoint con anotaciones. Se resaltan los campos generados a partir de los modelos Pydantic:

```

1 {
2     "pages": [
3         {
4             "index": 0,
5             "markdown": "...",
6             "images": [
7                 {
8                     "id": "img-0.jpeg",
9                     "top_left_x": 425,
10                    "top_left_y": 598,
11                    "bottom_right_x": 1283,
12                    "bottom_right_y": 893,
13                    "image_annotation": "{\n                        \"image_type\": \"image\", \n                        \"description\": \"A 3D rendering of a complex geometric model showing multiple intersecting planes and vertices. The model appears to be a wireframe or semi-transparent surface representing a mathematical or scientific visualization. The background is dark, making the white lines of the model stand out. The overall shape is somewhat abstract and organic in form.\"\n                    }\n                ],
16                 "dimensions": { ... }
17             }
18         ],
19         "model": "mistral-ocr-2503-completion",
20         "usage_info": { ... },
21         "document_annotation": "{\n            \"language\": \"en\",\n            \"summary\": \"The document introduces Mi\n22     }\n23

```

Se explica el formato obtenido:

#### `image_annotation`

- Es un *string* JSON (serializado) que respeta el esquema `Image` de Pydantic:
  - `image_type` : tipo de la caja (por ejemplo `"graph"`, `"table"`, `"image"`).
  - `description` : texto que describe o titula la figura extraída.

#### `document_annotation`

- Es un *string* JSON que sigue el esquema `Document` de Pydantic:

- **language** : código ISO-639-1 del idioma detectado.
- **summary** : resumen generado del contenido completo.
- **authors** : lista de nombres de autores extraídos.

Por diseño:

- **document\_annotation\_format** aplica al documento **completo**, así que la API devuelve un **único** campo **document\_annotation** en el **nivel raíz del JSON**.
- **bbox\_annotation\_format** aplica a cada caja detectada, y por eso en cada elemento de **pages[i]["images"]** aparece su propio **image\_annotation**.

En la respuesta final se tendrá:

```

1 {
2   "pages": [ ... ],
3   "model": "...",
4   "usage_info": { ... },
5   "document_annotation": "{ ... }"    ← SOLO UNA VEZ
6 }
```

Y dentro de cada página:

```

1 "pages": [
2   {
3     "index": 0,
4     "images": [
5       {
6         "id": "img-0.jpeg",
7         "image_annotation": "{ ... }"    ← UNA VEZ POR IMAGEN CADA VEZ QUE APARECE UN BBOX
8       }
9     ]
10   }
11 ]
12 ]
```

Así, **document\_annotation** se devuelve **una vez**, mientras que **image\_annotation** aparece **por cada imagen** detectada.

#### 6.4.3 Documento completo con Anotaciones

##### Realizar la Llamada a la API de OCR con Anotaciones

Se llama a la API de OCR de Mistral, especificando el modelo, las páginas a procesar, el documento en Base64, y los formatos de anotación para las cajas delimitadoras (bbox) y el documento. En este caso, se incluye la opción para obtener las imágenes de bbox.

```

1 annotations_response = client.ocr.process(
2   model="mistral-ocr-latest",
3   pages=list(range(8)), # Document Annotations has a limit of 8 pages, we recommend splitting yo
4   document={
```

```

5     "type": "document_url",
6     "document_url": f"data:application/pdf;base64,{base64_pdf}"
7   },
8   bbox_annotation_format=response_format_from_pydantic_model(Image),
9   document_annotation_format=response_format_from_pydantic_model(Document),
10  include_image_base64=True
11 )

```

## Definir la Función para Reemplazar Imágenes y Anotaciones en Markdown

Se define una función llamada `replace_images_in_markdown_annotated` que reemplaza los marcadores de posición de las imágenes en el texto Markdown con imágenes codificadas en Base64 y sus anotaciones correspondientes.

```

1 def replace_images_in_markdown_annotated(markdown_str: str, images_dict: dict) -> str:
2 """
3 Replace image placeholders in markdown with base64-encoded images and their annotation.
4
5 Args:
6     markdown_str: Markdown text containing image placeholders
7     images_dict: Dictionary mapping image IDs to base64 strings
8
9 Returns:
10    Markdown text with images replaced by base64 data and their annotation
11 """
12 for img_name, data in images_dict.items():
13     markdown_str = markdown_str.replace(
14         f"!([{img_name}]({img_name})", f"!([{img_name}]({data['image']}))\n\n**{data['annotation']}"
15     )
16 return markdown_str

```

## Definir la Función para Combinar Texto, Anotaciones e Imágenes en Markdown

Se define otra función llamada `get_combined_markdown_annotated` que combina el texto OCR, las anotaciones y las imágenes en un solo documento Markdown.

```

1 def get_combined_markdown_annotated(ocr_response: OCRResponse) -> str:
2 """
3 Combine OCR text, annotation and images into a single markdown document.
4
5 Args:
6     ocr_response: Response from OCR processing containing text and images
7
8 Returns:
9     Combined markdown string with embedded images and their annotation
10 """
11 markdowns: list[str] = ["**" + ocr_response.document_annotation + "**"]
12 # Extract images from page
13 for page in ocr_response.pages:
14     image_data = {}
15     for img in page.images:
16         image_data[img.id] = {"image":img.image_base64, "annotation": img.image_annotation}
17     # Replace image placeholders with actual images
18     markdowns.append(replace_images_in_markdown_annotated(page.markdown, image_data))
19
20 return "\n\n".join(markdowns)

```

## Mostrar el documento combinado renderizado en Markdown con anotaciones

Finalmente, se muestra el documento Markdown combinado que incluye tanto el texto extraído, las imágenes y sus anotaciones. Para las anotaciones, se define:

- **Anotación del documento** al inicio del documento.
- **Anotación cajas delimitadoras (bbox)** debajo de cada imagen/gráfico extraído.

```
1 display(Markdown(get_combined_markdown_annotated(annotations_response)))
```

```
{ "language": "en", "summary": "The document introduces Mistral 7B, a 7-billion-parameter language model designed for superior performance and efficiency. It outperforms the best open 13B model (Llama 2) across all evaluated benchmarks and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. The model uses grouped-query attention (GQA) for faster inference and sliding window attention (SWA) to handle sequences of arbitrary length with reduced inference cost. Mistral 7B is released under the Apache 2.0 license and includes a fine-tuned version, Mistral 7B - Instruct, which surpasses Llama 2 13B - chat model on human and automated benchmarks. The document also discusses the model's architectural details, performance on various benchmarks, and its efficiency in balancing performance and computational costs.", "authors": [ "Albert Q. Jiang", "Alexandre Sablayrolles", "Arthur Mensch", "Chris Bamford", "Devendra Singh Chaplot", "Diego de las Casas", "Florian Bressand", "Gianna Lengyel", "Guillaume Lample", "Lucile Saulnier", "Lélio Renard Lavaud", "Marie-Anne Lachaux", "Pierre Stock", "Téven Le Scao", "Thibaut Lavril", "Thomas Wang", "Timothée Lacroix", "William El Sayed" ] }
```

### Mistral 7B

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Téven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed



```
{ "image_type": "image", "description": "A 3D rendering of the text 'Mistral AI' in a gradient of warm colors, transitioning from orange to brown." }
```

#### Abstract

We introduce Mistral 7B, a 7-billion-parameter language model engineered for superior performance and efficiency. Mistral 7B outperforms the best open 13B model (Llama 2) across all evaluated benchmarks, and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. Our model leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length with a reduced inference cost. We also provide a model fine-tuned to follow instructions, Mistral 7B - Instruct, that surpasses Llama 2 13B - chat model both on human and automated benchmarks. Our models are released under the Apache 2.0 license. Code: <https://github.com/mistralai/mistral-src> Webpage: <https://mistral.ai/news/announcing-mistral-7b/>

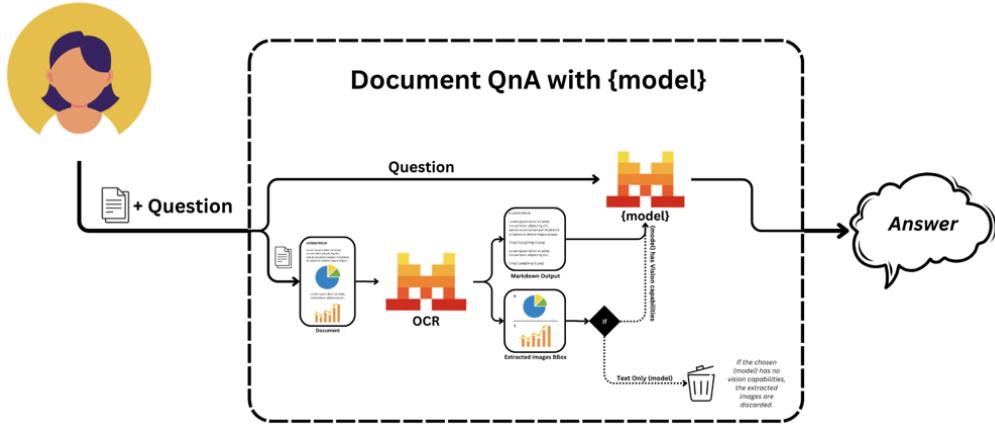
Visualización formateada (renderizado) del texto e imágenes extraídos mediante Markdown con anotaciones.

Todo el flujo en el notebook “Annotations for Structured Outputs and Data Extraction”:

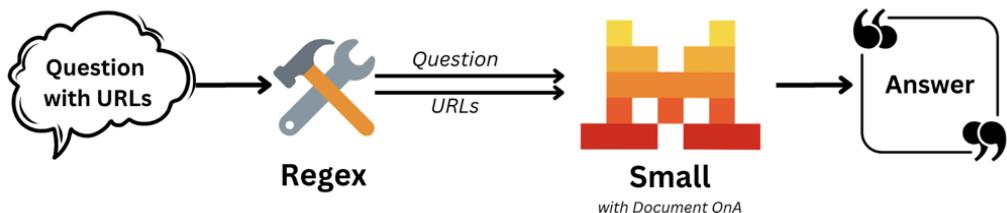
```
cookbook/mistral/ocr/data_extraction.ipynb at main · mistralai/cookbook
```

## 6.5 Document QnA Cookbook

Este feature usa el OCR que transforma documentos de texto e imágenes en texto puro y Markdown para luego usar el LLM de Mistral para comprender documentos(ya sea un PDF, una foto o una captura de pantalla, mediante URL) de forma fiable, eficiente y rentable.



como ejemplo practico mandaremos algunas preguntas con URLs para que sean analizados



## Creación del Cliente

para conectarse a la API de Mistral

```

1 from mistralai import Mistral
2
3 api_key = "API_KEY"
4 client = Mistral(api_key=api_key)
5 text_model = "mistral-small-latest"

```

## System prompt

El system prompt que recibirá el LLM.

```

1 system = "You are an AI Assistant with document understanding via URLs. You may be provided with U

```

## URLs

Usaremos Regex para extraer las URLs de las consultas del usuario.

```

1 import re
2
3 def extract_urls(text: str) -> list:
4     url_pattern = r'\b((?:https?|ftp)://(?:www\.)?[^s/$.?#].[^\s]*\b'
5     urls = re.findall(url_pattern, text)
6     return urls
7
8 # Example

```

```
9 extract_urls("Hi there, you can visit our docs in our website https://docs.mistral.ai/, we cannot
```

Ejemplo de salida: ['https://docs.mistral.ai']

Ejemplos de Prompts para el caso práctico.

- Could you summarize what this research paper talks about?  
[https://arxiv.org/pdf/2410.07073](https://arxiv.org/pdf/2410.07073.pdf)
- Explain this architecture: <https://arxiv.org/abs/2401.04088>

## Uso del feature

se crea el mensaje que se manda al LLM en la variable `user_message_content` luego se llama a `client.chat.complete(...)` con los mensajes y parámetros.

Finalmente obtenemos una respuesta contextual basada en el contenido extraído.

```
1 import json
2
3 messages = [{"role": "system", "content": system}]
4 while True:
5     user_input = input("User > ")
6     if user_input.lower() == "quit":
7         break
8
9     # Extract URLs from the user input, assuming they are always PDFs
10    document_urls = extract_urls(user_input)
11    user_message_content = [{"type": "text", "text": user_input}]
12    for url in document_urls:
13        user_message_content.append({"type": "document_url", "document_url": url})
14    messages.append({"role": "user", "content": user_message_content})
15
16    # Send the messages to the model and get a response
17    response = client.chat.complete(
18        model=text_model,
19        messages=messages,
20        temperature=0
21    )
22    messages.append({"role": "assistant", "content": response.choices[0].message.content})
23
24    print("Assistant >", response.choices[0].message.content)
```

Ejemplo de Ejecución y Salida:

User > Could you summarize what this research paper talks about? <https://arxiv.org/pdf/2410.07073>  
Assistant > The research paper titled "Pixtral 12B" introduces a 12-billion-parameter multimodal language model named Pixtral 12B. This model is designed to understand both natural images and documents, achieving leading performance on various multimodal benchmarks while also excelling in text-only tasks. Unlike many open-source models, Pixtral 12B does not compromise on natural language performance to excel in multimodal tasks.

### Key Features and Contributions:

1. \*\*Multimodal Capabilities\*\*:

- Pixtral 12B is trained to handle both images and text, making it versatile for a wide range of applications.
- It uses a new vision encoder trained from scratch, allowing it to ingest images at their natural resolution and aspect ratio, providing flexibility in the number of tokens used to process an image.

2. \*\*Performance\*\*:

- The model outperforms other open models of similar sizes (e.g., Llama-3.2 11B and Qwen-2-VL 7B) and even larger models like Llama-3.2 90B on various multimodal benchmarks.
- It also performs well on text-only benchmarks, making it a suitable drop-in replacement for both text and vision tasks.

3. \*\*Benchmarks and Evaluation\*\*:

- The paper introduces a new open-source benchmark, MM-MT-Bench, for evaluating vision-language models in practical scenarios.
- The evaluation protocol includes 'Explicit' prompts that specify the required output format, ensuring fair and standardized evaluation.

4. \*\*Architectural Details\*\*:

- Pixtral 12B is based on the transformer architecture and consists of a multimodal decoder and a vision encoder.
- The vision encoder is designed to handle variable image sizes and aspect ratios, using techniques like RoPE-2D for relative position encoding.

5. \*\*Applications\*\*:

- The model can be used for reasoning over complex figures, multi-image instruction following, chart understanding and analysis, and converting hand-drawn website interfaces into executable HTML code.

6. \*\*Open-Source Release\*\*:

- Pixtral 12B is released under the Apache 2.0 license, making it accessible for further research and development.

### Conclusion:

Pixtral 12B represents a significant advancement in multimodal language models, offering strong performance across both text and vision tasks. Its versatility and open-source nature make it a valuable tool for researchers and developers in the field of AI and machine learning.

User > quit

## Ejemplo con un archivo local

```
1 ruta_imagen = "RUTA/IMAGEN_LOCAL/IMAGEN.png"
2 # Cargamos el archivo
3 uploaded_pdf = client.files.upload(
4     file={
5         "file_name": ruta_imagen,
6         "content": open(ruta_imagen, "rb"),
7     },
8     purpose="ocr"
9 )
10 signed_url = client.files.get_signed_url(file_id=uploaded_pdf.id)
11
12 # Definimos el mensaje para el Chat
13 messages = [
14     {
15         "role": "user",
16         "content": [
17             {
18                 "type": "text",
19                 "text": "what is the last sentence/table in the document and replay the last table"
20             },
21             {
22                 "type": "image_url",
23                 # pasamos el url de la imagen cargada
24                 "image_url": signed_url.url
25             }
26         ]
27     }
28 ]
29
30 # Obtenemos respuesta
```

```

31 chat_response = client.chat.complete(
32     model=model,
33     messages=messages
34 )
35
36 print(chat_response.choices[0].message.content)

```

The last sentence in the document is:

"El pago de la penalidad no libera a EL CLIENTE de asumir los daños y perjuicios que pudieran exceder el m

The last table in the document is:

PROYECTO	ESPECIALIDADES
Levantamiento	Expediente de Levantamiento Asbuilt de sistemas de protección contra incendios existentes.
Asesoria Proyecto de Protección Contra Incendios	Criterio de Diseño Detección y Alarma de Incendios Agua Contra Incendios
Proyecto de Ingenierías Complementarias	Ingeniería Complementarias - Estudio Mecánica de Suelos - Arquitectura (Cuarto de Bombas) - Estructural (Cuarto de Bombas y Losa Cisterna) - Sanitaria (Cuarto de Bombas) - Mecánica (Cuarto de Bombas) Ingeniería Complementarias - Eléctrica (Cuarto de Bombas y Detección) Sistema contra Incendio (16 hrs) - NFPA 20 - NFPA 13 - NFPA 14 - NFPA 24
Capacitación	

(imagen original)

EC-24-GCO-CO-000161-0  
Los compromisos asumidos por EL CLIENTE en la presente cláusula tendrán una vigencia de veinticuatro (24) meses calendario contados a partir del día siguiente de terminada:

- (i) la vigencia del presente Contrato;
- (ii) la relación laboral entre el empleado y ESSAC, por la razón que fuere.

En caso de incumplimiento de EL CLIENTE a lo prescrito en la presente Cláusula, le será aplicable una penalidad de 10 Unidades Impositivas Tributarias (10 UIT), misma cantidad que deberá ser pagada dentro de los 05 cinco días calendario siguientes a ser remitida la notificación de incumplimiento y su requerimiento de pago por ESSAC. El pago de la penalidad no libera a EL CLIENTE de asumir los daños y perjuicios que pudieran exceder el monto de la penalidad.

#### 11. HONORARIOS

Los honorarios profesionales de **Engineering Services & Consulting S.A.C.**, para el desarrollo de lo entregables descritos en el punto 5 del presente convenio, corresponden a S/ 147,560.00 (CIENTI CUARENTA Y SIETE MIL QUINIENTOS SESENTA con 00/100 SOLES), según se indica en la siguiente tabla:

PROYECTO	ESPECIALIDADES	TOTAL
Levantamiento	Expediente de Levantamiento Asbuilt de sistemas de protección contra incendios existentes.	9,700.00
Asesoria Proyecto de Protección Contra Incendios	Criterio de Diseño Detección y Alarma de Incendios Agua Contra Incendios	16,500.00
Proyecto de Ingenierías Complementarias	Ingeniería Complementarias - Estudio Mecánica de Suelos - Arquitectura (Cuarto de Bombas) - Estructural (Cuarto de Bombas y Losa Cisterna) - Sanitaria (Cuarto de Bombas) - Mecánica (Cuarto de Bombas) Ingeniería Complementarias - Eléctrica (Cuarto de Bombas y Detección) Sistema contra Incendio (16 hrs) - NFPA 20 - NFPA 13 - NFPA 14 - NFPA 24 - NFPA 72	48,000.00
Capacitación	Ingeniería Complementarias - Eléctrica (Cuarto de Bombas y Detección) Sistema contra Incendio (16 hrs) - NFPA 20 - NFPA 13 - NFPA 14 - NFPA 24 - NFPA 72	4,600.00
	<b>COSTO DIRECTO</b>	119,000.00
	<b>GG + UTILIDADES (24%)</b>	28,560.00
	<b>SUB TOTAL (SIN IMPUESTOS)</b>	147,560.00

## 6.6 OCR Cookbook

[cookbook/mistral/ocr/tool\\_usage.ipynb at main · mistralai/cookbook](#)

## 7. Otros tópicos (basado en experiencia)

### 7.1 Costos y rendimiento

- **Precio estándar:** 1 000 páginas / USD 1; Annotations: 1 000 etiquetas / USD 3.
- **Batch Inference:** hasta 2 000 páginas / USD 1 con procesamiento concurrente.
- **Latencia típica:** 1–2 s por página en OCR básico; escalable con concurrencia.

 Mistral Medium 3 State-of-the-art performance. Simplified enterprise deployments. Cost-efficient.  Input (/M tokens) \$0.4 Output (/M tokens) \$2  mistral-medium-latest 	 Document AI & OCR Introducing the world's best document understanding API.  OCR \$1 / 1000 pages Annotations \$3 / 1000 pages  mistral-ocr-latest 	 Mistral Small 3.2 SOTA. Multimodal. Multilingual. Apache 2.0.  Input (/M tokens) \$0.1 Output (/M tokens) \$0.3  mistral-small-latest 
---	--	--

### 7.2 Despliegue y escalabilidad

- **SaaS:** API en la Plataforma de Mistral AI y próximamente en partners como Vertex AI.
- **On-prem:** soporte previsto para entornos corporativos.

### 7.3 Preprocesamiento de imágenes

- Deskew, binarización, reducción de ruido para mejorar OCR en scans antiguos o fotos de baja calidad.

### 7.4 Seguridad y cumplimiento

- TLS en tránsito, cifrado en reposo, retención configurable en S3 u otros almacenes.

### 7.5 Monitorización y alertas

- Orquestar con Step Functions o Airflow, monitorizar con CloudWatch o Datadog para métricas y errores.