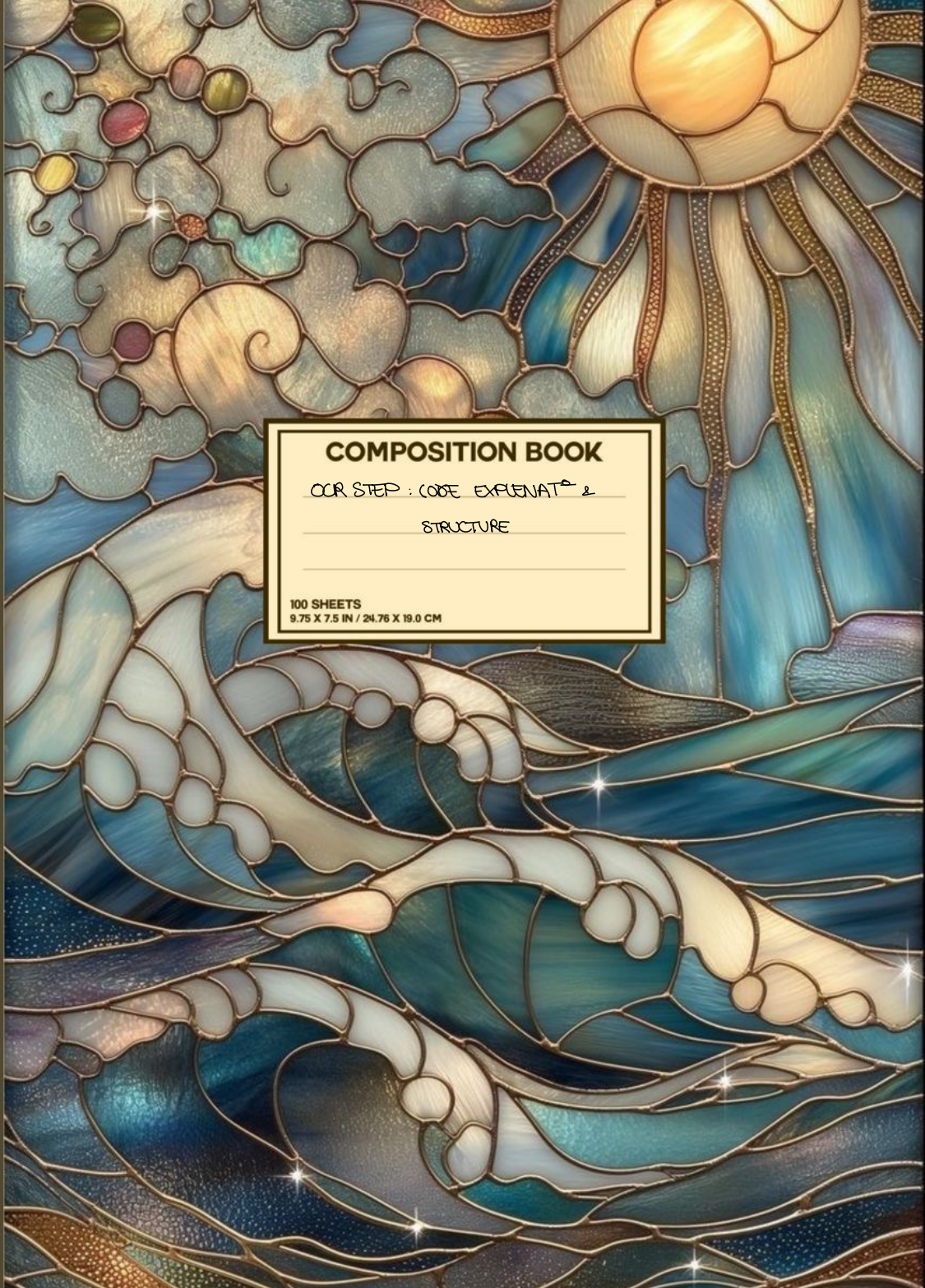
COMPOSITION BOOK
OCR STEP : CODE EXPLENATN &
STRUCTURE
100 SHEETS
9.75 X 7.5 IN / 24.76 X 19.0 CM

PROJECT STRUCTURE

CJCR/NewAttempt/

- main.py (main orchestrator)
- config.py (paths & settings common to all programs)
- ModelTraining.py
  - __init__.py
  - fix_labels.py (fix labels when they're names don't match the image's).
  - traing_classifier.py (training model w/ our dataset)
  - YOLOIDDetector.py (YOLO Detection class)
- processors/ => OCR processing pipeline.
  - __init__.py
  - contentDetector.py (detect content type: text, image, ID, audio, ...)
  - image_processor.py (image enhancement)
  - ocr_processor.py (text extraction)
  - pdf_loader.py (PDF to image).
  - text_processor.py (text formatting)
  - visualizer.py

STEP BY STEP

① MAIN.PY

- workflow: start → load document → for each page → [Process Pipeline] → Save Results → Finish
- INPUT: folder path containing PDF Document
  OUTPUT: folder w/ extracted text + visualization for each PDF
- key functions → process_pdf_file() - process single PDF
  → process_pdf_folder() - process all PDFs in folder
  → OutputManager class - Manage file organization

② CONFIG.PY

- Central location for all files & settings. Contains:
  - model paths
  - Input/Output folders
  - Processing Parameters
  - ModelTraing Settings
- key Variables → TRAINED_MODEL_PATH = "/path/to/trained_model.pt" (our trained model, for now only hosted on my PC)
  → INPUT_FOLDER = "/path/to/input/PDFs"
  → OUTPUT_FOLDER = "/path/to/output/results"

DON'T RUN IT !

③ MODEL_TRAINING / YOLOIDDETECTOR.PY

- workflow: Initialize → load Model → Detect Objects → Return Results
- Input : B&W data (images here)
  Output: List of Detected objects w/ Bounding Boxes, classes, confidence.
- key Components → preprare_dataset() - Prepare training data in YOLO format (YALN) & split dataset for training: 80/20
  → train_model() - Train model w/ parameters & data augmentation
  → detect_id_cards() - Detect ID cards & pictures in images.

③2 TRAINING_CLASSIFIER.PY

- workflow: fix labels → Prepare Dataset → Train Model → Save Model
- Input: Raw Images + Label file
  Output: Trained Model weights (best.pt)

  } will call YOLOIDDetector.py !

③3 FIX_LABELS.PY

Fix mismatched label file names & class IDs.

- workflow: Find Labels → Rename to match images → Convert Class IDs → Save.
- e.g: 0e2f75c6-img_154.txt → img_154.txt, class 15 → 0

④ PROCESSORS / OCR PROCESSING PIPELINE - workflow: PDF → Page Image → Preprocess → Detect Content → OCR → format → visualize.

④.1 PDF_LOADER.PY ⇒ convert PDF pages in2 images.

- workflow: load PDF → Extract Page → Convert to OpenCV Image → Return
- input : PDF file path
  Output: OpenCV image (numpy array)

④.2 IMAGE_PROCESSOR.PY ⇒ Enhance images (↗ contrast) for ⊛ efficient OCR.

- workflow: Grayscale → Denoise → Enhance Contrast → Sharpen → Return
- Input: Raw Image
  Output: Enhanced grayscale image.

(4.3) CONTENT_DETECTOR.PY ⇒ Determine content type (3D Pic / Pic / Text)

- WORKFLOW: Load YOLO model → Detect Objects → Classify Content → Return Type

Outputs: "id_card"; "picture"; "text"

(4.4) OCR_PROCESSOR.PY ⇒ Extract text from images using EasyOCR

- WORKFLOW: Load EasyOCR → Process Image → Extract Text → Return Results
- Input: Enhanced GrayScale image

Output: list of (Bounding Box, text, confidence) tuples.

(4.5) TEXT_PROCESSOR.PY ⇒ Format OCR results based on content type

- WORKFLOW: Check Content Type → Format Accordingly → Return Structured Text
- E.g Outputs → 3D card: [3D CARD] \n Extracted text... → Text: Raw exctracted text...

→ Picture: [PICTURE] \n Extracted text...

(4.6) VISUALIZE.PY ⇒ create visualization for OCR results

- WORKFLOW: Draw Bounding Boxes → Create Side Panel → Combine → Save Image
- INPUT: Image + OCR result + content type

OUTPUT: Combined visualized image (original + text extract panel)

RUNNING CODE:

↳ training phase (one-time setup): fix_labels.py → train_model.py → YOLOIDDetector.py → trained_model.pt

↳ processing phase (daily use): main.p

- pdf_loader.py
- img_processor.py
- content_detector.py
- ocr_processor.py
- text_processor.py
- visualizer.py

DATA FLOW

: the only one's you actually click "run" for, they call the other .py mentioned!

PDF FOLDER
↓ main.py selects PDF
PDF FILE
↓ pdf_loader.py converts
PAGE IMAGES
↓ content_detector.py classifies
CONTENT TYPE + BOUNDING BOXES
↓ ocr_processor.py extracts
TEXT + CONFIDENCE SCORE
↓ text_processor.py formats
STRUCTURED TEXT
↓ visualizer.py creates
VISUALIZATION + TEXT FILE

↳ key technology:

- YOLOv8 - Object Detection
- EasyOCR - text extraction
- PyMuPDF (fitz) - PDF processing
- OpenCV - image processing
- NumPy - Numerical Operations
- PIL/Pillow - Image Manipulation

NOW INCLUDING SPEECH TO TEXT

↳ flujo:

archivo subido → Detector_Tipo → [PDF/PNG/JPG] → [MP4/AVI/MOV] → [MP3/WAV] → Resultados Unificados

- [PDF/PNG/JPG] → ¿? → OCR Pipeline (existente)
- [MP4/AVI/MOV] → ¿? → Extraer frames → OCR Pipeline
- [MP3/WAV] → ¿? → Speech-To-Text → Solo texto (no OCR)

↳ ESTRUCTURA ACTUALIZADA

```
NewAttempt /
├── main.py                    (Actualizar el Orchestrator)
├── config.py                  (Config Actualizado)
├── ModelTraining/
│   ...
│
├── processors/
│   ├── file_type_detector.py      NUEVO: Detectar q tipo de Archivo es
│   ├── audio_processor.py         NUEVO: Speech-To-Text
│   ├── video_processor.py         NUEVO: Extraer Frames
│   ├── pdf_loader.py              Existentes
│   ├── contentDetector.py         Existentes
│   ...                            Existentes
└── requirements.txt   ⟹ AÑADIR DEPENDENCIAS AUDIO/VIDEO.
```

These are just an idea of how to incorporate speech-to-text to current OCR code, NOT THE FINAL RESULT ¡¡!!

Paso 1: Actualizar config.py

```python
# Añadir al config.py
SUPPORTED_FILE_TYPES = {
    'pdf': ['application/pdf', '.pdf'],
    'image': ['image/png', 'image/jpeg', 'image/jpg', '.png', '.jpg', '.jpeg'],
    'video': ['video/mp4', 'video/avi', 'video/mov', '.mp4', '.avi', '.mov'],
    'audio': ['audio/mp3', 'audio/wav', 'audio/m4a', '.mp3', '.wav', '.m4a']
}

# Speech-to-text settings
WHISPER_MODEL = 'base'  # 'tiny', 'base', 'small', 'medium', 'large'
LANGUAGE = 'es'  # o 'en', 'auto' para detección automática
```

Paso 2: file_type_detector.py

```python
import os
import mimetypes
from config import SUPPORTED_FILE_TYPES

class FileTypeDetector:
    def __init__(self):
        mimetypes.init()

    def detect(self, file_path):
        """Detectar tipo de archivo"""
        if not os.path.exists(file_path):
            return None

        # Por extensión
        ext = os.path.splitext(file_path)[1].lower()

        # Por MIME type
        mime_type, _ = mimetypes.guess_type(file_path)

        # Determinar categoría
        for category, type_list in SUPPORTED_FILE_TYPES.items():
            if ext in type_list or (mime_type and mime_type in type_list):
                return category

        return None

    def get_processor_pipeline(self, file_type):
        """Obtener pipeline de procesamiento según tipo"""
        pipelines = {
            'pdf': ['pdf_loader', 'ocr'],
            'image': ['image_loader', 'ocr'],
            'video': ['video_extractor', 'image_loader', 'ocr'],
            'audio': ['speech_to_text']
        }
        return pipelines.get(file_type, [])
```

Paso 3: audio_processor.py

```python
import whisper
import tempfile
import os
from config import WHISPER_MODEL, LANGUAGE

class AudioProcessor:
    def __init__(self):
        print(f"Loading Whisper model: {WHISPER_MODEL}")
        self.model = whisper.load_model(WHISPER_MODEL)

    def transcribe(self, audio_path):
        """Convertir audio a texto usando Whisper"""
        try:
            result = self.model.transcribe(
                audio_path,
                language=LANGUAGE,
                fp16=False  # Usar CPU
            )

            return {
                'text': result['text'],
                'segments': result.get('segments', []),
                'language': result.get('language', 'unknown')
            }
        except Exception as e:
            print(f"❌ Error transcribing audio: {e}")
            return None

    def extract_from_video(self, video_path):
        """Extraer audio de video y transcribir"""
        try:
            import moviepy.editor as mp

            # Extraer audio del video
            video = mp.VideoFileClip(video_path)

            # Guardar audio temporalmente
            with tempfile.NamedTemporaryFile(suffix='.mwv', delete=False) as tmp:
                audio_path = tmp.name
                video.audio.write_audiofile(audio_path, verbose=False,
logger=None)

            # Transcribir
            result = self.transcribe(audio_path)

            # Limpiar
            os.unlink(audio_path)
            video.close()

            return result

        except ImportError:
            print("❌ Install moviepy: pip install moviepy")
            return None
        except Exception as e:
            print(f"❌ Error processing video: {e}")
            return None
```

WAIT FOR YASSIN'S CODE, NOT THIS ONE ¡¡!!

Paso 4: video_processor.py

```
import os
import cv
import tempfile
from datetime import datetime

class VideoProcessor:
    def __init__(self, fps=1):
        self.fps = fps  # Frames por segundo a extraer

    def extract_frames(self, video_path, output_dir=None):
        """Extraer frames del video para OCR"""
        frames = []
        timestamps = []

        if output_dir and not os.path.exists(output_dir):
            os.makedirs(output_dir)

        cap = cv2.VideoCapture(video_path)
        if not cap.isOpened():
            print(f"❌ Cannot open video: {video_path}")
            return frames, timestamps

        video_fps = cap.get(cv2.CAP_PROP_FPS)
        frame_interval = int(video_fps / self.fps) if self.fps > 0 else 1

        frame_count = 0
        extracted_count = 0

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # Extraer frame cada frame_interval
            if frame_count % frame_interval == 0:
                frames.append(frame)
                timestamp = frame_count / video_fps
                timestamps.append(timestamp)

                # Guardar frame si se especifica output_dir
                if output_dir:
                    timestamp_str = str(datetime.fromtimestamp(timestamp)).replace(':', '-')
                    frame_path = os.path.join(output_dir, f"frame_{extracted_count:04d}_{timestamp_str}.jpg")
                    cv2.imwrite(frame_path, frame)

                extracted_count += 1

            frame_count += 1

        cap.release()

        print(f"✅ Extracted {extracted_count} frames from video")
        return frames, timestamps

    def extract_keyframes(self, video_path, output_dir=None):
        """Extraer frames clave (cuando hay cambios significativos)"""
        frames = []

        cap = cv2.VideoCapture(video_path)
        prev_frame = None

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            if prev_frame is not None:
                # Calcular diferencia entre frames
                diff = cv2.absdiff(prev_frame, frame)
                non_zero = cv2.countNonZero(cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY))

                # Si hay cambio significativo, guardar frame
                if non_zero > 10000:  # Umbral ajustable
                    frames.append(frame)

            prev_frame = frame

        cap.release()
        return frames
```

Paso 5: Actualizar main.py

```
#!/usr/bin/env python3
"""
Main OCR Pipeline - Soporte para PDF, imágenes, audio, video
"""

import os
import signal
import sys
from datetime import datetime
import cv2

os.environ['PYTORCH_MPS_DISABLE'] = '1'
os.environ['CUDA_VISIBLE_DEVICES'] = ''

sys.path.append(os.path.dirname(os.path.abspath(__file__)))

from config import INPUT_FOLDER, OUTPUT_FOLDER
from processors.pdf_processor import PDFProcessor, DocTypeDetector
from processors.pdf_cache import PDFCache
from processors.image_preprocessor import ImagePreprocessor
from processors.content_detector import ContentDetector
from processors.ocr_processor import OCRProcessor
from processors.text_postprocessor import TextPostprocessor
from processors.visualizer import Visualizer
from processors.audio_processor import AudioProcessor
from processors.video_processor import VideoProcessor

stop_processing = False

def signal_handler(sig, frame):
    global stop_processing
    print("\n🛑 Stopping...")
    stop_processing = True

signal.signal(signal.SIGINT, signal_handler)

class MultiFormatProcessor:
    """Procesador para múltiples formatos de archivo"""
    def __init__(self):
        self.file_detector = FileTypeDetector()
        self.pdf_loader = PDFLoader()
        self.preprocessor = ImagePreprocessor()
        self.content_detector = ContentDetector()
        self.ocr_processor = OCRProcessor()
        self.postprocessor = TextPostprocessor()
        self.visualizer = Visualizer()
        self.audio_processor = AudioProcessor()
        self.video_processor = VideoProcessor(fps=1)  # 1 frame por segundo

    def process_file(self, file_path, output_base_folder):
        """Procesar cualquier tipo de archivo"""
        file_type = self.file_detector.detect(file_path)
        filename = os.path.basename(file_path)

        print(f"📄 Processing: {filename} ({file_type})")

        if file_type == 'image':
            return self._process_image(file_path, output_base_folder)
        elif file_type == 'pdf':
            return self._process_video(file_path, output_base_folder)
        elif file_type == 'audio':
            return self._process_audio(file_path, output_base_folder)
        elif file_type == 'video':
            print(f"❌ Unsupported file type: {file_type}")
            return None

    def _process_pdf(self, pdf_path, output_base_folder):
        """Procesar PDF (existente)"""
        # Tu código existente para PDFs
        pass

    def _process_image(self, image_path, output_base_folder):
        """Procesar imagen individual"""
        image = cv2.imread(image_path)
        if image is None:
            print(f"❌ Cannot read image: {image_path}")
            return None

        processed_image = self.preprocessor.process(image)
        content_type = self.content_detector.detect_content_type(processed_image)
        ocr_results = self.ocr_processor.extract_text(processed_image)
        extracted_text = self.postprocessor.process(ocr_results, content_type)

        # Crear carpeta de salida
        output_folder = os.path.join(output_base_folder, os.path.splitext(os.path.basename(image_path))[0])
        output_folder = os.path.join(output_base_folder, base_name)
        os.makedirs(output_folder, exist_ok=True)

        # Guardar resultados
        result_file = os.path.join(output_folder, f"{base_name}_extracted.txt")
        with open(result_file, 'w', encoding='utf-8') as f:
            f.write(f"Archivo: {base_name}\n")
            f.write(f"Content Type: {content_type}\n")
            f.write("="*60 + "\n\n")
            f.write(extracted_text)

        # Crear visualización
        viz_image = self.visualizer.create(processed_image, ocr_results, content_type, extracted_text)
        viz_path = os.path.join(output_folder, f"{base_name}_visualization.png")
        cv2.imwrite(viz_path, viz_image)

        print(f"✅ Processed image: {result_file}")
        return result_file

    def _process_video(self, video_path, output_base_folder):
        """Procesar video: Speech-to-text + OCR en frames"""
        base_name = os.path.splitext(os.path.basename(video_path))[0]
        output_folder = os.path.join(output_base_folder, base_name)
        os.makedirs(output_folder, exist_ok=True)

        all_text = f"Video Analysis: {base_name}\n"
        all_text += "="*60 + "\n\n"

        # 1. Speech-to-text del audio del video
        print("🎤 Extracting speech from video...")
        audio_text = self._extract_audio_text(video_path)

        if audio_text:
            all_text += "AUDIO TRANSCRIPTION:\n"
            all_text += "="*60 + "\n"
            all_text += audio_text['text'] + "\n\n"

        # 2. OCR en frames del video
        print("🎞️ Extracting frames for OCR...")
        frames_folder = os.path.join(output_folder, "frames")
        os.makedirs(frames_folder, exist_ok=True)

        frames, timestamps = self.video_processor.extract_frames(video_path, frames_folder)

        all_text += "VISUAL CONTENT (OCR from frames):\n"
        all_text += "="*60 + "\n"

        for i, (frame, timestamp) in enumerate(zip(frames, timestamps)):
            if stop_processing:
                break

            print(f"  Frame {i+1}/{len(frames)} ({int(timestamp)}s)")

            # Process frame con OCR
            processed = self.preprocessor.process(frame)
            content_type = self.content_detector.detect_content_type(processed_frame)
            ocr_results = self.ocr_processor.extract_text(processed_frame)
            frame_text = self.postprocessor.process(ocr_results, content_type)

            if frame_text.strip():
                all_text += f"\n[Frame at {timestamp:.2f}s - Content_Type]\n"
                all_text += frame_text + "\n"

        # Guardar resultados
        result_file = os.path.join(output_folder, f"{base_name}_analysis.txt")
        with open(result_file, 'w', encoding='utf-8') as f:
            f.write(all_text)

        print(f"✅ Processed video: {result_file}")
        return result_file

    def _process_audio(self, audio_path, output_base_folder):
        """Procesar audio: Solo speech-to-text"""
        base_name = os.path.splitext(os.path.basename(audio_path))[0]
        output_folder = os.path.join(output_base_folder, base_name)
        os.makedirs(output_folder, exist_ok=True)

        print("🎤 Transcribing audio...")
        result = self.audio_processor.transcribe(audio_path)

        if not result:
            print("❌ Failed to transcribe audio")
            return None

        # Guardar resultados
        result_file = os.path.join(output_folder, f"{base_name}_transcription.txt")
        with open(result_file, 'w', encoding='utf-8') as f:
            f.write(f"Archivo: {base_name}.audio\n")
            f.write(f"Detected Language: {result.get('language', 'unknown')}\n")
            f.write("="*60 + "\n\n")
            f.write(result['text'] + "\n\n")

            # Opcional: Incluir segmentos con timestamps
            if 'segments' in result:
                f.write("\n\nSEGMENTS WITH TIMESTAMPS:\n")
                f.write("="*60 + "\n")
                for segment in result['segments']:
                    start = segment['start']
                    end = segment['end']
                    text = segment['text']
                    f.write(f"[{start:.2f}s - {end:.2f}s] {text}\n")

        print(f"✅ Transcribed audio: {result_file}")
        return result_file

def process_folder(input_folder, output_base_folder):
    """Procesar todos los archivos en una carpeta"""
    print(f"📂 Processing files from: {input_folder}")

    processor = MultiFormatProcessor()
    supported_extensions = ['.pdf', '.png', '.jpg', '.jpeg', '.mp4', '.avi', '.mov', '.mp3', '.wav', '.m4a']

    # Recopilar todos los archivos soportados
    all_files = []
    for root, dirs, files in os.walk(input_folder):
        for file in files:
            if any(file.lower().endswith(ext) for ext in supported_extensions):
                all_files.append(os.path.join(root, file))

    if not all_files:
        print("❌ No supported files found")
        return

    print(f"📁 Found {len(all_files)} supported files")

    processed = 0
    for file_path in all_files:
        if stop_processing:
            break

        try:
            result = processor.process_file(file_path, output_base_folder)
            if result:
                processed += 1
                print(f"✅ Completed: {os.path.basename(file_path)}")
        except Exception as e:
            print(f"❌ Error processing {file_path}: {e}")

    print(f"\n🎉 Processed {processed}/{len(all_files)} files successfully")

if __name__ == "__main__":
    print("🚀 Starting Multi-Format OCR & Speech-to-Text Pipeline")
    print(f"📂 Input: {INPUT_FOLDER}")
    print(f"📂 Output: {OUTPUT_FOLDER}")
    print("⏸️ Press Ctrl+C to stop\n")

    process_folder(INPUT_FOLDER, OUTPUT_FOLDER)
    print("✅ Processing complete!")
```

add in requirement.txt:

openai-whisper >= 20231117

moviepy >= 1.0.3

SpeechRecognition >= 3.10.0

pydub >= 0.25.1